IEC 63504-2804

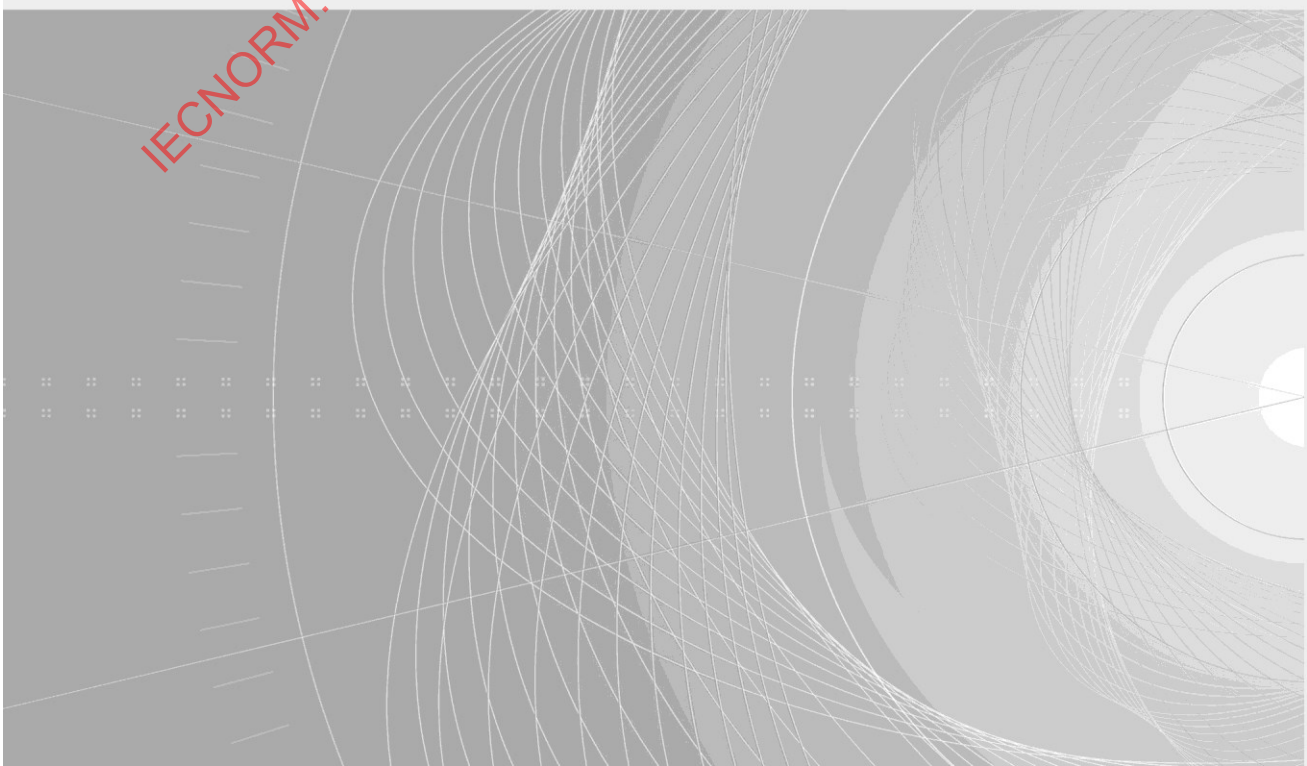
Edition 1.0 2023-10

INTERNATIONAL STANDARD

IEEE Std 2804™

colour inside

Software-Hardware Interface for Multi-Many-Core



IEC 63504-2804:2023-10(en) IEEE Std 2804-2019

IEC Secretariat
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Tel.: +41 22 919 02 11
info@iec.ch
www.iec.ch

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

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

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

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 It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished
Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and once a month by email.

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

IEC Products & Services Portal - products.iec.ch
Discover our powerful search engine and read freely all the publications previews. With a subscription you will always have access to up to date content tailored to your needs.

Electropedia - www.electropedia.org
The world's leading online dictionary on electrotechnology, containing more than 22 300 terminological entries in English and French, with equivalent terms in 19 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC 63504-2804

Edition 1.0 2023-10

INTERNATIONAL STANDARD

IEEE Std 2804™

colour inside

Software-Hardware Interface for Multi-Many-Core

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

ICS 25.040.01, 35.060

ISBN 978-2-8322-7514-6

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

Contents

1. Overview 10
1.1 Scope 10
1.2 Purpose 10
1.3 Word usage 10
1.4 General introduction 11

2. Normative references 18

3. Definitions 18

4. SHIM concepts 18
4.1 Topology—ComponentSet 18
4.2 Inter-core communication—CommunicationSet 19
4.3 Frequency and voltage—FrequencyVoltageSet 20
4.4 Communication network utilization and contention—ContentionGroupSet 21
4.5 Performance 22
4.6 Power—PowerConfiguration 26
4.7 Vendor extensions 26
4.8 Configuration 27

5. Roadmap 29
5.1 General 29
5.2 Further componentization of SHIM XML 30
5.3 Hardware-related software properties 30
5.4 Schema refinement for smaller XML 30

6. SHIM interface 31
6.1 shim20.xsd 31
6.2 Conventions 40
6.3 Enumeration 40
6.4 Shim 42
6.5 SystemConfiguration 42
6.6 ComponentSet 43
6.7 FrequencyVoltageSet 53
6.8 AddressSpaceSet 56
6.9 CommunicationSet 59
6.10 ContentionGroupSet 63
6.11 PowerConfiguration 65
6.12 VendorExtension 66

7. Use cases 68
7.1 Performance estimation: Auto-parallelizing compiler 68
7.2 Tool configuration—RTOS configuration tool 70
7.3 Hardware modeling 71

8. SHIM XML authoring rules and guidelines 71
8.1 File name [rule] 72
8.2 The naming of various objects [rule] 73
8.3 Level of detail and precision [guideline] 73

9. Common Configuration File (CCF) 73

9.1 Concept 73
9.2 Interface 75
9.3 Examples 79

10. FAQ 80

Annex A Participants 83

SOFTWARE-HARDWARE INTERFACE FOR MULTI-MANY-CORE

FOREWORD

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

   IEEE Standards documents are developed within IEEE Societies and Standards Coordinating Committees of the IEEE Standards Association (IEEE SA) Standards Board. IEEE develops its standards through a consensus development process, 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 IEEE and serve without compensation. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards. Use of IEEE Standards documents is wholly voluntary. *IEEE documents are made available for use subject to important notices and legal disclaimers (see* https://standards.ieee.org/ipr/disclaimers.html *for more information).*

   IEC collaborates closely with IEEE in accordance with conditions determined by agreement between the two organizations. This Dual Logo International Standard was jointly developed by the IEC and IEEE under the terms of that agreement.

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

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

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

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

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

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

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

9) Attention is drawn to the possibility that implementation of this IEC/IEEE Publication may require use of material covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. IEC or IEEE shall not be held responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patent Claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility.

IEC 63504-2804/IEEE Std 2804 was processed through IEC technical committee 91: Electronics assembly technology, under the IEC/IEEE Dual Logo Agreement. It is an International Standard.

The text of this International Standard is based on the following documents:

IEEE Std	FDIS	Report on voting
2804 (2019)	91/1873A/FDIS	91/1889/RVD

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

The language used for the development of this International Standard is English.

The IEC Technical Committee and IEEE Technical Committee have decided that the contents of this document will remain unchanged until the stability date indicated on the IEC website under webstore.iec.ch in the data related to the specific document. At this date, the document will be

- reconfirmed,
- withdrawn, or
- revised.

IMPORTANT – The "colour inside" logo on the cover page of this document indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

IEEE Standard for Software-Hardware Interface for Multi-Many-Core

Developed by the

Design Automation Standards Committee
of the
IEEE Computer Society

Approved 7 November 2019

IEEE SA Standards Board

Abstract: This standard is intended primarily for tool developers and hardware developers who would use Software Hardware Interface for Multi-Many-core (SHIM) to exchange hardware description for software tools. It also attempts to provide software developers with insights into what hardware information is described in SHIM to foster understanding of the intention and the extent of SHIM.

Keywords: IEEE 2804, many-core, multicore, SHIM, software hardware interface, software tools

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page, appear in all standards and may be found under the heading "Important Notices and Disclaimers Concerning IEEE Standards Documents." They can also be obtained on request from IEEE or viewed at http://standards.ieee.org/ipr/disclaimers.html.

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

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

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

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

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

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

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

Translations

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

Official statements

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

Comments on standards

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

Comments on standards should be submitted to the following address:

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

Laws and regulations

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

Copyrights

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

Photocopies

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

Updating of IEEE Standards documents

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

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

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

Errata

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

Patents

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

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

IEEE Introduction

This introduction is not part of IEEE Std 2804-2019, IEEE Standard for Software-Hardware Interface for Multi-Many-Core.

This document is intended primarily for tool developers and hardware developers who would use SHIM to exchange hardware description for software tools. It also attempts to provide software developers with insights into what hardware information is described in SHIM to foster understanding of the intention and the extent of SHIM.

This document begins with the introduction to SHIM, providing the background, the overall concept and model. It is followed by a clause detailing the concept of SHIM, such as the purpose, scope, design, interface, limitation, providing the basic idea why SHIM is as specified in this document and also trying to explain the basic principles for future extension of the specification. A clause describing the interface follows, which is a description of SHIM XML schema and APIs that are mostly derived directly from the schema via XML data binding technique. A clause providing some of the detailed use cases follows, allowing the reader to gain insights into how SHIM can be used in action. Finally, this document ends with various annexes that provide further detailed information.

IEEE Standard for Software-Hardware Interface for Multi-Many-Core

1. Overview

1.1 Scope

The scope of this standard includes performance estimation accuracy for complex processors like Very Long Instruction Word (VLIW) core and complex contention scenarios, description of caches to include uncached memory regions and caches for subsets of memories, properties for coarse power consumption estimation, and reusability by separating eXtensible Markup Language (XML) files for processor description and other memory/communication-related information.

1.2 Purpose

The Software-Hardware Interface for Multi-Many-core (SHIM) defines an architecture description standard from the software design perspective—this provides a common interface that abstracts the hardware properties that are critical to enable multicore tools.

1.3 Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (shall equals is required to).[1, 2]

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (should equals is recommended that).

The word *may* is used to indicate a course of action permissible within the limits of the standard (may equals is permitted to).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (can equals is able to).

[1]The use of the word *must* is deprecated and cannot be used when stating mandatory requirements, *must* is used only to describe unavoidable situations.
[2]The use of *will* is deprecated and cannot be used when stating mandatory requirements, *will* is only used in statements of fact.

1.4 General introduction

Multicore processors have become the norm, and processors with tens and even more than a hundred cores are emerging. These multicore and many-core processors vary not only in the number of cores, but also in inter-connects, cluster organization, and memory systems (including hierarchy and cache coherency), among others. While the trend for an increasing number of cores is both natural and unavoidable from a processor design perspective, this poses tremendous challenges to the software developers to cope with the significant hardware variance, while bearing a burden to re-use the existing and newly created software for different hardware. Moreover, all this must occur while achieving the performance expected from the multicore processors, which requires a deep understanding of the specific multicore architecture. Various tools, such as auto-parallelizing compilers, parallelization tools, OS/middleware configurators, and performance analysis tools, aid developers to design, implement, and analyze the software. However, these tools must comprehend the complex multicore processor, transferring the burden to the tool developers. Therefore, lowering the cost of supporting new multicore hardware by various tools is critical, but there has been a lack of effort in academia or industry to solve these issues, thereby hindering the development of the multicore tool eco-system.

The SHIM, Software-Hardware Interface for Multi-many-core, is a joint industrial and academic effort to standardize the interface between the multicore hardware and the software tools. As a result, the aim is to lower the cost of supporting new multicore hardware using the standard interface. This will encourage the development of new innovative multicore tools, resulting in a richer eco-system of multicore technologies, which in turn should benefit system developers, semiconductor vendors, and tool vendors.

1.4.1 Interface

The SHIM is defined as an XML schema. A multicore hardware implementation is expressed as a set of SHIM XML files that can be used by various tools (see Figure 1).

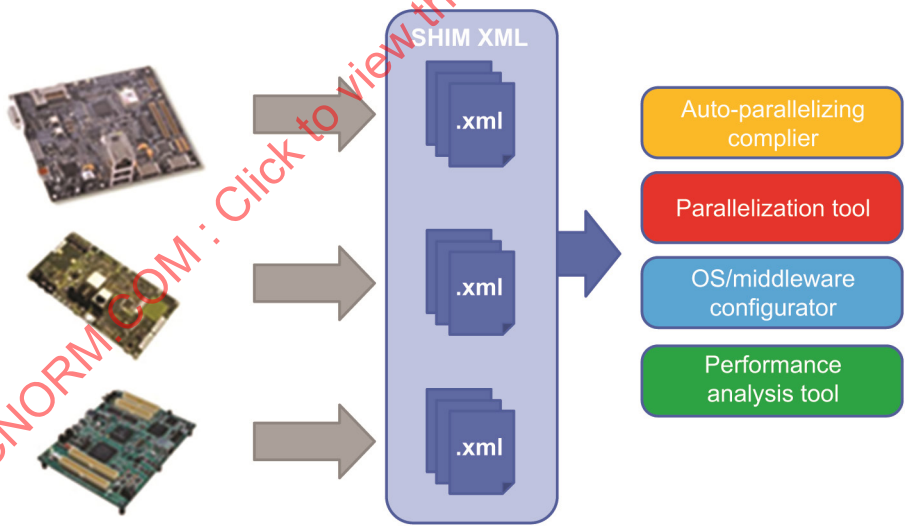


Figure 1—SHIM provides the interface between the hardware and the software tools

The SHIM XML file has a tree structure in which XML elements are used to describe the architectural and platform properties of a multi-many-core system. Figure 2 illustrates the main SHIM XML elements. A system description starts with a *SystemConfiguration* element with up to five children components, namely *ComponentSet*, *AddressSpaceSet*, *CommunicationSet*, *FrequencyVoltageSet,* and *ContentionGroupSet* each containing further child elements. The *ComponentSet* contains *MasterComponent (*representing a processor or accelerator) and *SlaveCodemponent (*representing a memory block or memory subsystem). The *AddressSpaceSet* contains one or more *AddressSpace*, which in turn contains *SubSpace*. The

CommunicationSet contains any number of *Communication* elements, describing the connection and communication between a pair of *MasterComponents*. The *FrequencyVoltageSet* contains definitions of frequency and voltage domains that defines sets of components sharing the same clocks or input voltage. It also contains definitions of component operating points, through *OperatingPointsSet*. Finally, the *ContentionGroupSet* allows for the defining of *ContentionGroup*s, which are communication resources used by component-to-component communications. Using those *ContentionGroup*s allows users to determine communication resources utilization and potential contentions.

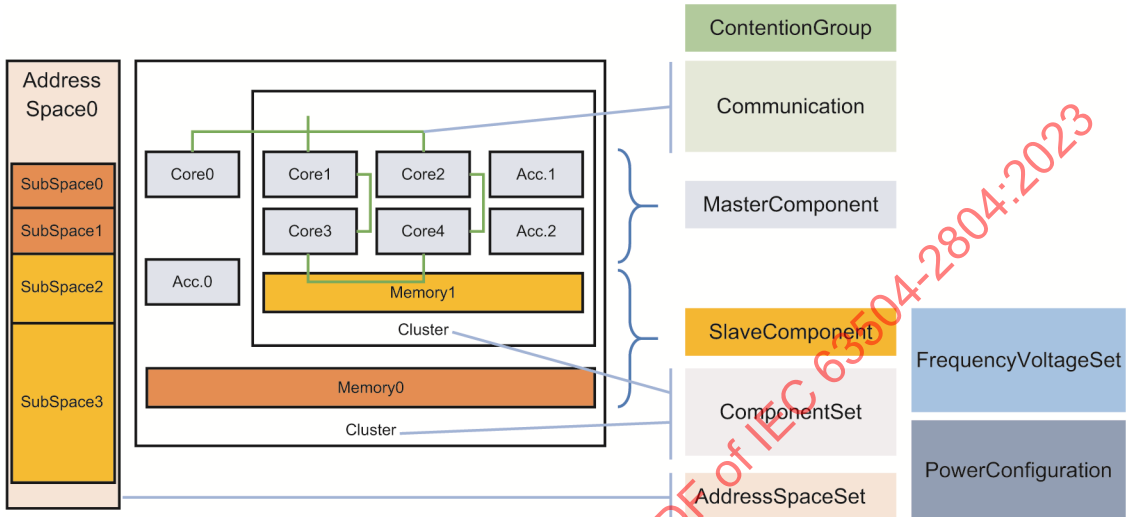


Figure 2—The SHIM elements mapped to a pseudo-multicore hardware

A *ComponentSet* can nest itself. For example, it can be used to express a chip that contains multiple hardware clusters, each cluster containing multiple cores with a cluster local memory. It can also be used to describe a board, which in turn may contain one or more multicore chips. A *ComponentSet* can even be used to describe a system with multiple boards, each board connected via PCI Express, for example. As such, the *ComponentSet* tree describes the multicore hardware system topology. This topological architectural information is important for software tools to be able to identify the number of cores, the location of the memory devices, and how cores are organized into different clusters.

Since SHIM is for software tools, it is essential to understand from a software perspective, the connection and communication mechanism between the cores (including accelerators), as well as how these cores can access the different memories. The former is described as *CommunicationSet* containing different communication classes. A simple example of defined classes is *InterruptCommunication*, which contains one or more "connection" classes, which binds a pair of *MasterComponents*. For memory access, the *SubSpace* contained in the *AddressSpace* includes its start address and size and one or more *MasterSlaveBinding*, containing references to a *MasterComponent* and *SlaveComponent*, describing which core/accelerator can access which memory through the address range.

The hardware architectural information described so far allows tools to understand the hardware topology, and how the cores and memory devices are connected. However, this alone is often insufficient for many tools, since the application software supported by these tools must not just 'run', but run with performance qualifiers. To achieve this, the tools must 'estimate' the rough performance so that the system designers and software developers know the expected performance from the given application and multicore hardware. Therefore, SHIM, in addition to the hardware topological information, describes the performance properties associated with the processor cycles consumed to perform the various core-to-core communication (*CommunicationSet*) and also the memory access cycles by different cores and accelerators. The performance is described as *Performance* element, which contains *Latency* and *Pitch*, expressed in processor cycles. The *Performance* element exists for each *CommunicationSet*, for each specific pair of two

MasterComponents. For memory access performance, for each *MasterSlaveBinding* of each *SubSpace*, and for each *AccessType*, which are defined for each *MasterComponent*, a specific *Performance* element is included. So, for each different access type (e.g., read or write, word access, or double word access), a different *Performance* element is provided. The cycles can be described in a form of the triplet, which is 'best', 'typical', and 'worst', to accommodate the possible performance variance. The tool must be intelligent enough to benefit from these figures, such as analyzing the application code if it is issuing a sequential memory access, which generally falls into the use of the 'best' cycles. Note that the cycles mentioned here are processor-cycles, whose related frequency is defined using *OperatingPoint*s.

1.4.2 Software view—what is in and what is not

Tools should primarily use SHIM to aid developing software that runs on multi-many-core hardware. Therefore, the key strategy in defining the SHIM specification is to describe the hardware but only for the information that is relevant to such tools. This is referred to as a 'software view' of hardware, as opposed to 'hardware view', where the focus would be the physical/electrical means of inter-connects between processing cores, the Network on Chip (NoC) protocol used to route the memory read request by a particular core, the number of processor pipeline stages, the cache coherency protocol, etc., unless these features matter greatly to some class of tools that aid software development.

It is tempting and relatively easy to include additional hardware properties in SHIM; however, this will result in a more complex SHIM XML, requiring more effort to grasp the schema and complicating the effort for tools to use this information. Furthermore, the most critical issue is the challenge to create a SHIM XML in the first place—leading to limited adoption of the SHIM standard.

The basic principle is to capture the properties that affect the software at the architectural design level. This is to say, if a design-aid tool uses SHIM to produce an appropriate software design for a particular hardware described by a SHIM XML, then the design should not require modification at the software architectural level at the later stages of system development.

Although the "software architectural design level" is the baseline, it is sometimes difficult to agree on whether a particular hardware property is important. The rule of thumb is that if an actual (even imaginable) use case cannot be derived, the SHIM specification excludes it.

For various reasons, a number of potential hardware properties have not been included in the current specification. One of the primary reasons is that the excluded types of hardware properties are peripheral to existing properties in the specification. Such hardware properties may be included in a future version of the specification, but it was decided to take an evolutionary approach and stabilize the more basic properties first.

The most basic properties selected for inclusion in SHIM are the following: topology, address space, inter-core communication, and performance and configuration.

With SHIM 2.0, several optional additions have been included in the interface, namely *FunctionalUnitSet*, to define in which functional unit an instruction is executed and improve estimation accuracy, *ContentionGroup* to handle communication resource utilization and contention, which may impact software execution performance, *OperatingPoints,* and *PowerConfiguration*, which may influence software implementation strategy or automatic optimization.

Clause 5 of this document illustrates how the information provided by a SHIM XML file can be exploited by software development tools. Three examples are provided: performance estimation, tool configuration, and hardware modeling.

1.4.3 XML

The SHIM interface uses extensible markup language (XML); specifically, the SHIM XML uses the XML Schema (XSD) to define its XML structure. XSD is essentially the same as a UML class diagram. Each unique hardware is described by one or more SHIM XML files. There must be at least one file for the system description. Optionally there could be separate files for core instruction set descriptions. The standard also allows for the defining of additional files for power configuration and for vendor extensions. All files must conform to the SHIM XML schema. The UML class diagram representation of SHIM XML schema is shown in Figure 3.

The XML schema allows the definition of the SHIM XML structure, but with the help of a validating parser that reads XML files, the schema also allows validation of SHIM XML. Validating parsers are readily available, both openly and commercially, often bundled with various XML related libraries in many different programming languages.

Therefore, technically speaking, the SHIM XML schema, or the shim20.xsd, is the core interface definition of SHIM.

1.4.3.1 Data binding

A common technique to read XML files is via SAX or DOM libraries. Using XSD, it is possible to generate class libraries in many choices of programming languages by running a schema compiler against the shim.xsd. The generated library includes all the SHIM XML classes of the chosen programming language, with automatically added methods or functions to get and set the data. This allows tools to access hardware properties expressed in a SHIM XML similar to accessing normal objects in their programming languages.

1.4.3.2 Who creates SHIM XML

The hardware provider is expected to create and provide the SHIM XML, which will then be used by the software tools. On the other hand, a hardware provider may not provide the SHIM XML. If SHIM XML can only be authored by the hardware provider, it can be a significant roadblock in the hardware's adoption. Therefore, reference authoring tools (see 1.4.5) are made freely available along with the specification. If a user has access to the basic technical reference manuals, and either simulator or actual hardware (e.g., evaluation board), the reference authoring tools allow for the creation of the SHIM XML for most multi-many-core hardware in fewer than 1–2 days.

1.4.4 SHIM Editor

Although a SHIM XML schema is relatively simple, as can be seen on the UML Class diagram representation of the SHIM XML schema (see Figure 3), the resulting SHIM XML file can be quite large, mostly due to all the *Performance* element descriptions for all types of memory accesses. Writing it manually can be tedious and error-prone, so an editor tool called the SHIM Editor has been developed to foster authoring a SHIM XML file. The generated SHIM XML file is shown in Figure 4, and the SHIM Editor prototype's main window is shown in Figure 5.

1.4.5 Reference authoring tools

In addition to the specification itself, SHIM also provides a free set of reference authoring tools. As a reference, anyone can provide their own version of the SHIM authoring tools. The OpenSHIM[3] provides the reference authoring tool, SHIM Editor, for the following reasons:

a) Easy authoring of SHIM XML to enable better adoption

b) Serves as a sample SHIM application with source code

[3] See https://github.com/openshim/shim.

1.4.6 Changes introduced in SHIM 2.0

The following changes have been introduced in SHIM 2.0:

— SHIM 2.0 introduces processor functional units modeling with the new *FunctionalUnitSet* element, which allows for the defining of the functional units of a processor and their supported instructions. See 6.6.7 and 6.6.8 for more information.

— SHIM 2.0 improves the SHIM XML file componentization and reuse by allowing to define a processor model in a separate file, through the use of the *FunctionalUnitSetFile.* See 6.6.8.

— The *Instruction* element has been extended with new parameters allowing, for instance, to define the instruction bit width, the number of input and output, the supported signedness, the instruction SIMD width, etc. The complete parameter list and more information can be found in 6.6.10.

— Instruction modeling has been further extended by allowing to define custom instructions using the *CustomInstruction* element. This element provides support for complex instructions. More information can be found in 6.6.11.

— SHIM 2.0 replaces the previous *ClockFrequency* element, which was limited to clocks definition, with the new *OperatingPointSet* element that allows for the defining of one or more operating points (consisting in a frequency value and an optional voltage value) for system components. This enables modeling of dynamic frequency scaling and/or dynamic voltage scaling for systems that support it. See 4.3 for more information.

— SHIM 2.0 introduces frequency and voltage domains, which allow defining sets of components sharing resp. the same frequencies or the same voltages. See 4.3 for more information.

— SHIM 2.0 allows improves cache hierarchy definition by allowing the definition of more complex cache hierarchies, like for instance memory-side caches. *MasterComponent*s now refer to their related *Cache*s using the *CacheRef* element. See 6.6.3 for more information.

— SHIM 2.0 further improves cache modeling by extending the *Cache* element with new parameters such as replacement policy, write back/write allocation policy, prefetch support, etc. The complete parameter list and more information can be found in 6.6.3.

— In SHIM 2.0, cached regions can be explicitly defined by referencing the cache element caching the memory accesses in a *PerformanceSet* element. See 6.8.7.

— SHIM 2.0 introduces support for communication resource utilization and contention modeling with the new element *ContentionGroup* and *ContentionGroupSet*. See 4.4 for more information.

— SHIM 2.0 introduces the new *PowerConfiguration* element, which enables power consumption modeling by defining the system components' power consumption. To further improve componentization and flexibility, *PowerConfiguration* is an optional element defined in a separate file. See 4.6 for more information.

— SHIM 2.0 introduces the new *VendorExtension* element, which provides a solution for vendor desiring to extend a SHIM model with their own functionalities. To further improve componentization and flexibility, *VendorExtension* is an optional element defined in a separate file. See 4.7 for more information.

— SHIM 2.0 enforces a stricter XML schema to improve interoperability. A SHIM 2.0 XML file only allows the new *shim* element as the root element. This root element only accepts a *SystemConfiguration*, a *FunctionalUnitSet*, a *PowerConfiguration* or a *VendorExtension* child element. See Clause 6 for more details.

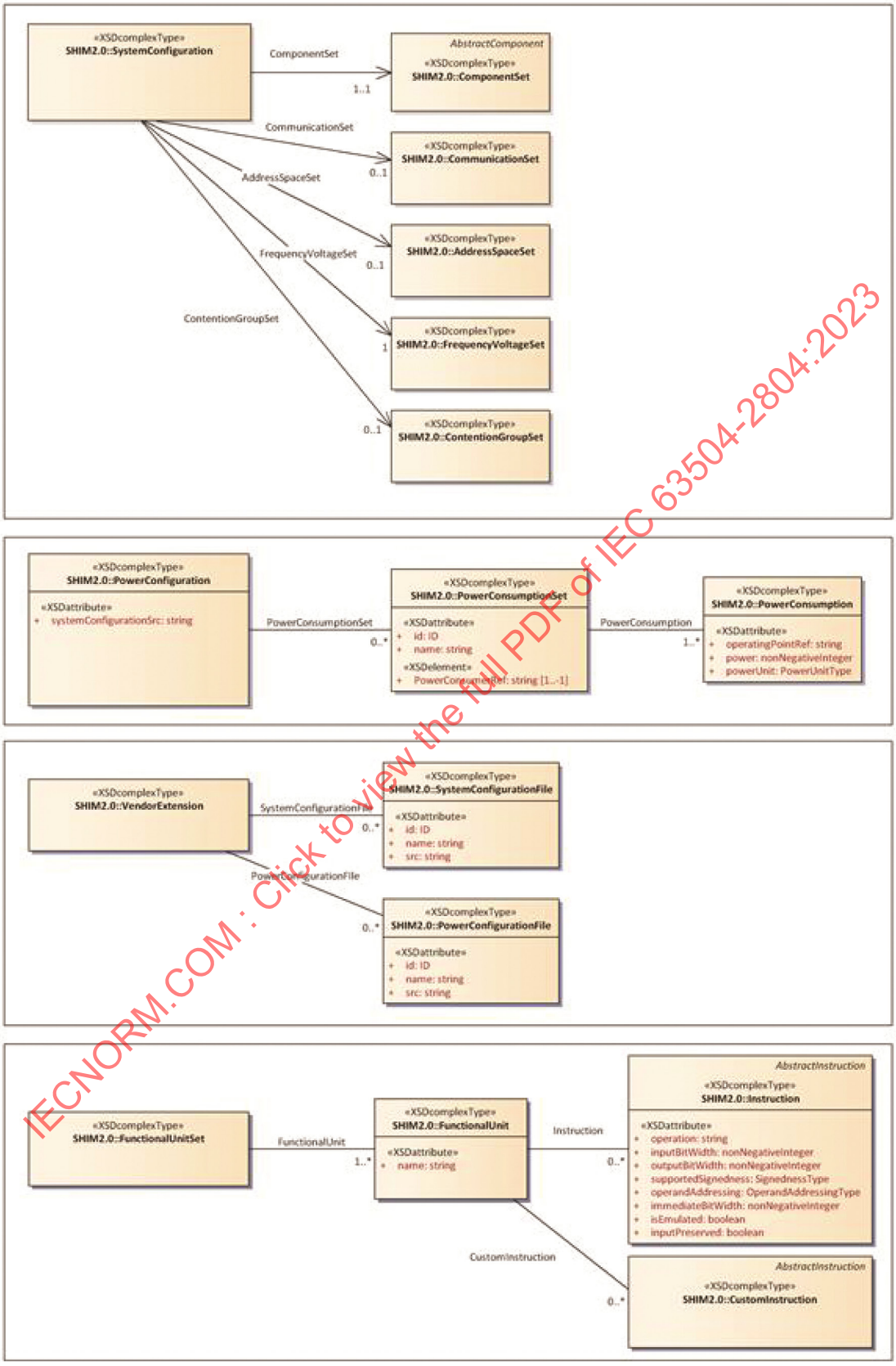


Figure 3—Class diagram representation of the SHIM XML schema (top-level elements)

Figure 4—SHIM XML file example

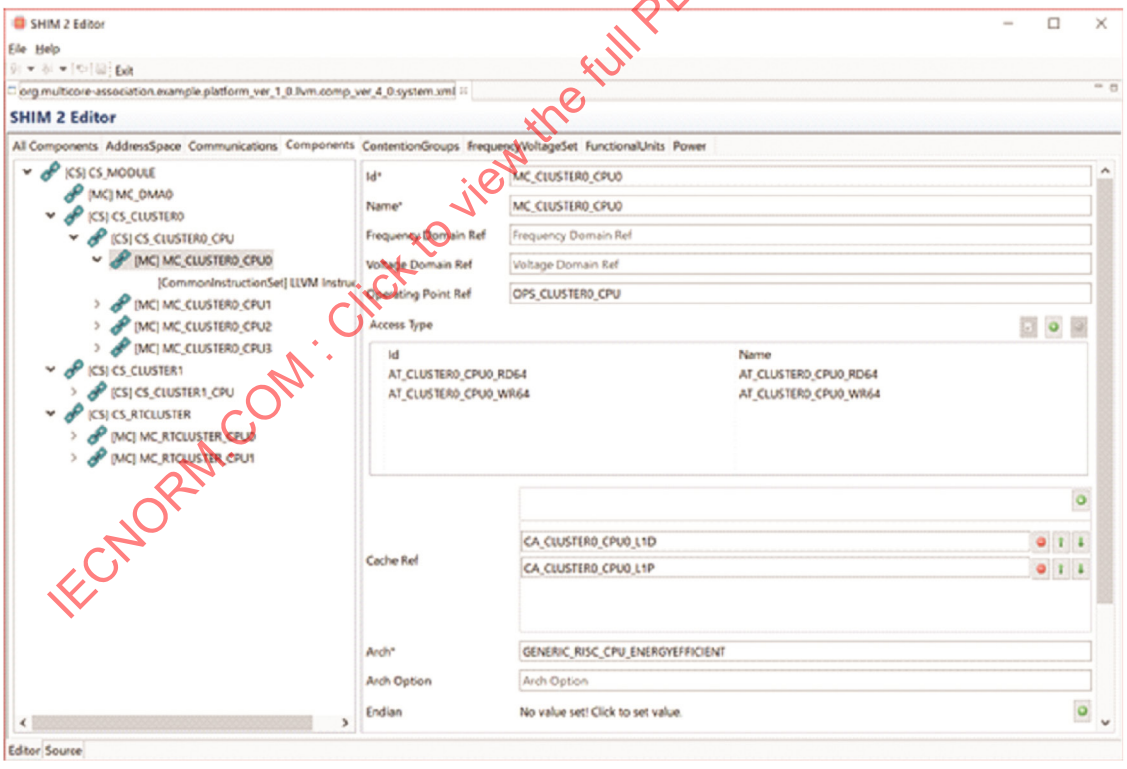


Figure 5—SHIM editor main window

2. Normative references

Not applicable

3. Definitions

The *IEEE Standards Dictionary Online* should be consulted for terms.[4]

4. SHIM concepts

This clause describes the major SHIM concepts, providing the basic idea why SHIM is as specified in this document, and also attempts to indicate the principle for future extension of the specification. This chapter should provide a foundation for understanding the SHIM interface (see Clause 6), so it is strongly recommended to read this thoroughly before diving into the interface details.

4.1 Topology—ComponentSet

A simple hardware setup may consist of a single processor core and a single memory—however, the multi-many-core hardware has multiple processor cores and memory devices of various types in various configurations. The combination and configuration of processor and memory characterize the multi-many-core hardware, and it is essential for software tools to comprehend them.

SHIM expresses the particular mix of processors and memory devices as 'topology'. In the electrical circuits' terminology, topology "*is the form taken by the network of interconnections of the circuit components. Different specific values or ratings of the components are regarded as being the same topology. Topology is not concerned with the physical layout of components in a circuit, nor with their positions on a circuit diagram. It is only concerned with what connections exist between the components. There may be numerous physical layouts and circuit diagrams that all amount to the same topology.*"[5] From the SHIM's perspective, the topology is extended further. In addition to processor cores and memory devices, which are components in the electrical terminology, *'clusters'* are also included, which is a particular set or grouping of processor cores and memory devices. Usually there are electrical connections between a cluster and other hardware elements; however, SHIM does not necessarily deal with actual electrical connections, so the cluster may not form any connection. However, it is critical for software tools to see how processor cores and memory devices are grouped as it is often an indication of a performance difference, therefore SHIM includes *cluster* as a part of its topological expression.

A *cluster* is composed of any combination of another (inner) cluster, processor core, and a memory device. SHIM has its own way of classifying and naming these objects (Table 1). A processor core is represented as a *MasterComponent* object. As can be seen from the table, a *MasterComponent* can also be some type of accelerator (e.g., a DMA controller). The objective of MasterComponent is to represent those electrical components that play the role of the master component in the traditional master-slave bus setup, but only if they are relevant to the software view SHIM defines.

[4] *IEEE Standards Dictionary Online* is available at: http://dictionary.ieee.org.
[5] http://en.wikipedia.org/wiki/Topology_(electronics).

Table 1—SHIM representation of hardware components

SHIM term	Hardware term
ComponentSet	A cluster of any level (a hardware board itself is also a cluster)
MasterComponent	Processor core, accelerator, or other master devices
SlaveComponent	Memory

The *cluster*, or *ComponentSet*, can be used to express not only a processor core cluster but also a hardware board. It can also be extrapolated to represent a system composed of multiple boards—in this case, the outermost cluster is the system boundary itself.

Memory—AddressSpaceSet A software program accesses memory through a logical window called the address space. Processor hardware usually supports multiple address spaces, for different access privileges, for example. An address space is further subdivided into multiple subspaces or address blocks. When a program makes an access somewhere in a memory device, it performs this by issuing a load or store instruction with its source or destination address falling into any one of the subspaces. To accommodate this memory setup, SHIM has a group of objects called *AddressSpaceSet*. An *AddressSpaceSet* can contain multiple *AddressSpace*, and each *AddressSpace* can contain multiple *SubSpace*.

A *SubSpace* is mapped to a physical memory device, or *SlaveComponent*, residing in some cluster, or *ComponentSet*. To describe the binding for which *SlaveComponent* is mapped to a specific *SubSpace*, the SHIM specification uses an object called *MasterSlaveBinding*. The object describes the mapping between a memory device and a memory subspace; it also indicates which *MasterComponent* (e.g., a processor core) has access to the memory. Since it is possible for multiple *MasterComponents* to have access to a memory *SubSpace*, a set object called *MasterSlaveBindingSet* is also defined to group multiple *MasterSlaveBinding* objects.

By exploring the objects under the *AddressSpaceSet*, a tool can discover what memory spaces are available and which processor core or accelerator has what kind of access to those. Multiple *AddressSpace*/*SubSpace* may share the same *SlaveComponent*. If the sharing occurs for only parts of the physical memory, it can be divided into multiple *SlaveComponents*.

4.2 Inter-core communication—CommunicationSet

For software to run on multiple processor cores and accelerators with some degree of cooperative manner, it often exchanges data, which may be available via a shared memory region. The software must also trigger, synchronize, or perform mutual exclusion in some way. In cases where shared memory is not available, some form of core-to-core or *MasterComponent* to *MasterComponent* communications is required. To accommodate this situation, SHIM defines a class of objects called *CommunicationSet*. All SHIM objects have a child object called *ConnectionSet*, which includes one or more *Connection* that describes the source and destination *MasterComponents* for the communication. The variety of *ConnectionSet* classes have similar communication mechanisms (see Table 2).

Table 2—Inter-core communication classes

CommunicationSet classes	Description
SharedRegisterCommunication	Shared register based communication. Often such hardware provides a set of registers that can be accessed by multiple processor cores.
SharedMemoryCommunication	Shared memory based communications. An operation type is specified from TAS (Test and set), LLSC (Load-link/Store conditional), CAX (Compare and exchange), and OTHER (other unspecified operation).
EventCommunication	An event is often a register bitmap based communication—if a processor core raises an event (Boolean), that is sent to another core and can be seen as the mapped event signaled in its event register. It may or may not trigger an interrupt.
FIFOCommunication	A FIFO is sometimes used for inter-core communication and often implemented as FIFO registers, possibly with buffers of varying depth.
InterruptCommunication	This is a typical inter-processor-interrupt. This object only has the *ConnectionSet*.

Each class has its unique properties or attributes. All classes include connection information describing which pairs of cores are connected by the particular communication object. Since there can be multiple connections, the object contains *ConnectionSet*, which in turn contains any number of *Connection*. Each *Connection* contains references to a pair of *MasterComponents*.

Software tools can use this information to obtain the type of *MasterComponent*-to-*MasterComponent* communication mechanisms are supported by a particular hardware implementation represented by a SHIM XML. Note that the connection can be across multiple *ComponentSet* boundaries, even if it traverses the chip or hardware board boundaries.

4.3 Frequency and voltage—FrequencyVoltageSet

Software execution time strongly depends on the component clock frequencies. As such, specifying the clock frequencies of those components is necessary if one wants to estimate software performances, which is relevant for SHIM (see Clause 7).

To define the components' clock frequencies, SHIM 1.0 provided basic support through the *ClockFrequency* object. However, modern multi- and many-core platforms support advanced techniques such as dynamic frequency scaling, which allows changing the clock frequency of a component dynamically. These techniques, combined with dynamic voltage scaling, allows tuning the operating point of the platform components to optimize the power consumption while still providing the desired quality of service or meeting real-time execution deadlines. Nowadays, most of those techniques are unavoidable, e.g., to fit an application power budget or to improve the autonomy of autonomous battery-operated systems.

The software architecture and implementation must generally be adapted to take the most benefit of those techniques. It is therefore greatly relevant for SHIM to include some level of modeling of the dynamic frequency and voltage scaling capabilities of a platform. Consequently, SHIM 2.0 provides an improved support for the definition of component frequency and voltage operating points. The main classes associated with this feature are listed in Table 3.

Most platform clock trees are designed in such a way that components are partitioned in several frequency domains. Inside those domains, all components share the same clock frequency. For instance, in a CPU cluster, all the cores typically run at the same frequency. Similarly, the input voltage trees are designed in such a way that components are partitioned in voltage domains, where all components inside a domain share the same input voltage. There is, however, no direct correspondence between both domains and it is very often the case that all components inside a voltage domain do not belong the same frequency domain. For instance, in a platform, a GPU and a CPU cluster might share the same voltage input but run at

different clock frequencies. In SHIM 2.0, it is possible to model these complex configurations by defining *FrequencyDomain* and *VoltageDomain* objects, which allow to define separately groups of components sharing respectively the same clock frequency or the same input voltage.

In platforms supporting dynamic frequency and/or voltage scaling, the components will generally support several operating points, characterized by different clock frequencies and input voltages. To define those operating points, SHIM 2.0 provides the *OperatingPointSet* class, which contains a list of *OperatingPoint* objects. Each of those *OperatingPoint* objects defines a valid component operating point. This *OperatingPoint* must provide a supported clock frequency value and optionally the associated input voltage.

To associate a *FrequencyDomain*, a *VoltageDomain* and *OperatingPointSet* to any component, one should use respectively the component object's *frequencyDomainRef*, *voltageDomainRef*, and *operatingPointSetRef* attributes.

Table 3—Component frequency and voltage classes

FrequencyVoltageSet classes	Description
FrequencyDomain	A frequency domain that allows for the defining of a group of components that will share the same clock frequency.
VoltageDomain	A voltage domain that allows for the defining of a group of components that will share the same voltage input.
OperatingPointSet	An *OperatingPointSet* that allows for the defining of all the valid operating points supported by a *Component*.
OperatingPoint	An *OperatingPoint* defines the frequency and, optionally, the associated input voltage of a component operating point.

4.4 Communication network utilization and contention—ContentionGroupSet

Multi- and many-core platforms can be leveraged by distributing the computation tasks on the available computation resources. Those tasks must generally communicate data between them to keep executing the program, which they do by using various communication mechanisms (see 4.2 for more information on how SHIM model those aspects). To be able to perform the data transfer between the communicating components, a platform provides several communication resources such as busses, crossbars, network-on-chip, DMA, etc.

Those communication resources are limited in bandwidth as they can only accept a maximum amount of data at any time. Consequently, if several communication components end up using a communication resource simultaneously, they might reach the maximal bandwidth of this resource and create a communication contention that will slow down the communication time and potentially negatively impact software performances.

For instance, Figure 6 illustrates a situation where two cores try to write to the same memory through a shared bus. If both accesses are simultaneous, the bus bandwidth will be shared, each core getting half of the available bus bandwidth. Consequently, transfer times will be longer and software execution might also increase.

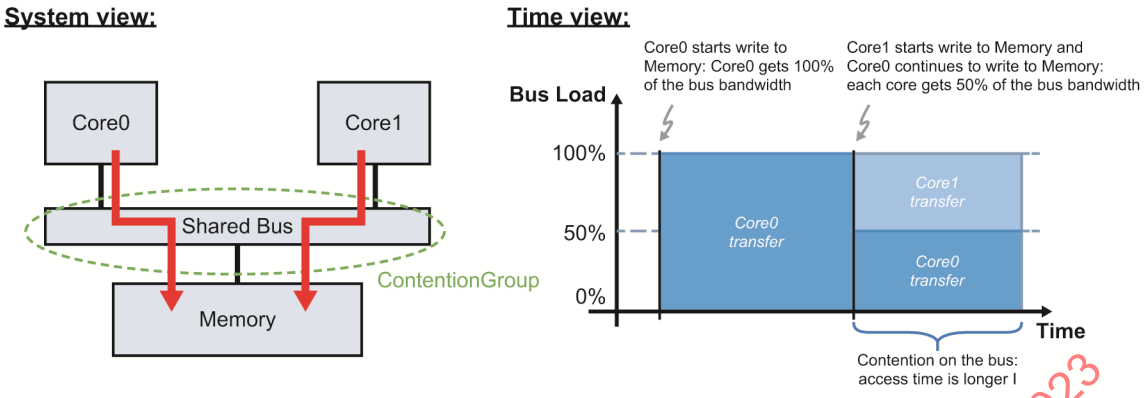


Figure 6—The contention in a shared bus

Previous versions of SHIM were not able to model those situations as it provided no way of modeling communication resource bandwidth and contention. To improve estimation accuracy, SHIM 2.0 provides the *ContentionGroupSet* class that allows for the defining of communication resources using *ContentionGroup* objects. To avoid modeling unnecessary details (see 1.4.2), a *ContentionGroup* does not care about the resource behavior, which allows ignoring complex aspects such as protocol, buffers, etc. A *ContentionGroup* only models a communication resource utilization and allows a vendor to specify its maximum bandwidth.

Each *ContentionGroup* explicitly lists all the accesses that make use of the modeled communication resource. This information can then be used (for instance by a performance estimation tool) to analyze and evaluate the communication resource utilization and potential contentions. To define this list, each *ContentionGroup* object contains one or more *PerformanceSetRef* objects, each one defining a reference to the *PerformanceSet* object making use of the communication resource.

Optionally, a *ContentionGroup* can be given a maximum bandwidth by using the *DataRate* and *Throughput* elements. The *DataRate* element allows for the defining of the bandwidth in terms of the amount of data per second, whereas the *Throughput* element allows for the defining of the bandwidth in terms of the amount of data per cycle. When using the *Throughput* attribute, it is therefore mandatory to define the related frequency domain as well.

In the example of Figure 6, a *ContentionGroup* could be defined to model the shared bus. Its bandwidth could then be specified using a *DataRate* or *Throughput* element, and the *PerformanceSetRef* of both accesses would be listed in the *ContentionGroup* object.

4.5 Performance

4.5.1 General

As expected, different processor hardware has different performance characteristics. The performance characteristics can be very complex for a multi-many-core hardware and will have a tremendous impact on the software design. Since SHIM's principle is to capture the properties that affect the software at the architectural design level, it is intrinsic to include such performance properties (see Table 4).

There are significant performance variations among the different processor, memory, and interconnect architectures, so all performance properties are expressed as a triplet of best, typical, and worst cycles. Some architectures are highly deterministic and may have little variation in the performance of some operations; this will be depicted with the triplet bearing similar, if not the same, values. The software tool

can use this information to determine the hardware dynamism or determinism by examining the deviation in the values. For such hardware, the estimation based on SHIM XML can be highly accurate, well under the 20% error rate that SHIM targets (even possibly nearing single digits of error percentage). Some hardware could have fairly dynamic performance characteristics, performing some operations mostly in two cycles, and possibly in 200 cycles in some cases, for example. This dynamic behavior often is derived from a wide range of speculative and probabilistic algorithms employed by modern hardware; this 'best-effort' approach, as opposed to a 'guarantee' approach, is quite popular and the trend continues.

SHIM, as said in 1.4.2, provides software with a simpler view of the underlying hardware and it avoids descriptions of what speculative algorithm is supported and its detailed spec. The triplet performance representation provides a window to adapt the dynamism by carefully setting up the three values, encapsulating the various hardware mechanism underneath. After all, it is technically infeasible, if not impossible, to achieve 100% accuracy in the hardware performance estimation—the idea is to obtain accurate enough performance estimation for system architectural design—the rest must be optimized in the later phase of system development. This approach is reasonable since the final set of software is unavailable before the system development stage and there are many other factors that influence the development, and thus the design, as the project progresses.

SHIM XML files are created for a specific hardware (and system software if necessary) configuration. If, for example, quality of service (QoS) is to impact the performance characteristics at a level greater than the goal of 20% error rate, multiple SHIM XML files must be authored or a Common Configuration File (CCF) (see Clause 9) must be used to describe the variation in performance.

Table 4—Performance properties in SHIM

Performance Property	Related SHIM Object	Description
Instruction execution	*CommonInstructionSet, Instruction*	The execution cycles of processor instructions. The instruction set is described as LLVM IR, and the cycles of a particular processor architecture are expressed in terms of these LLVM IR instructions. See CommonInstructionSet (6.6.6) for more information.
Memory access	*SubSpace, MasterSlaveBinding, Accessor, AccessType*	The processor cycles for accessing a memory. Each processor core can have different cycles for different access types such as read or write and accessing by byte, word, double word accesses, etc.
Inter-core communication	*CommunicationSet* classes	The time needed for a particular connection between two *MasterComponent* for a particular *Communication* class, such as *InterruptCommunication*, in processor cycles.

4.5.2 Latency and pitch

The performance object is characterized by a pair of triplets - one associated with 'latency', the other for the 'pitch' (see Figure 7). The latency, or *Latency* in terms of SHIM class, is specifically the processor cycles for performing the particular operation. The *Pitch* is a trickier process—it is the size of stride when executing the operation in a consecutive manner, also expressed in processor cycles. As indicated previously, modern hardware has a mechanism to speculate what would be the next software action. When accessing a memory, for example, the hardware has a cache that reads the memory in its line size, even if a smaller size of memory is requested by a particular 'load' instruction. In essence, it can read the next memory address ahead of time, hoping the next 'load' instruction will follow at the consecutive address (called a speculative fetch). If that fetch proves true, the next read operation can complete by reading from the cache, without actually accessing the slower main memory. The hardware supports other similar mechanisms—all trying to take advantage of repetitive software behavior. This action results in the performance characteristics that, if the similar operation is performed repeatedly in some way, the average

execution cycles per operation are less than it would be if it is not. The *Pitch* is specifically meant to describe this performance property.

The software tool's job is to see if a particular operation is repeated, and use the *Latency* and *Pitch* triplets accordingly.

4.5.3 Using triplets

Any factors that influence performance characteristics should be expressed in the triplet of 'best', 'typical', and 'worst' to describe the performance variations. This is described with examples in Table 5.

Please note that statistically speaking, the 'typical' value is not the average but the mode.

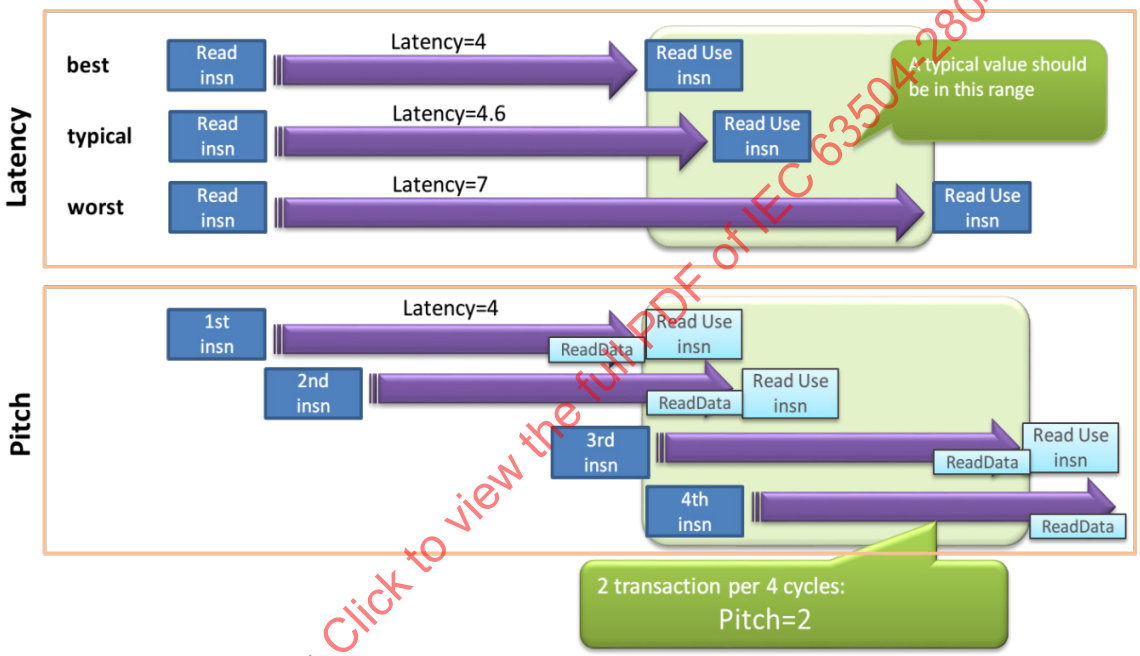
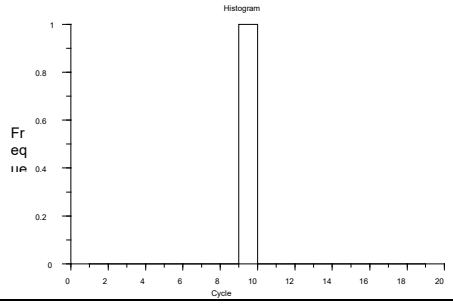
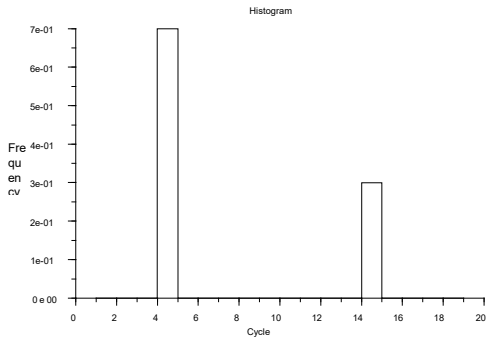
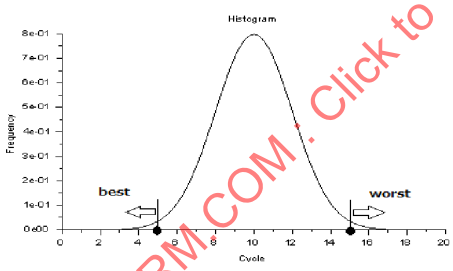
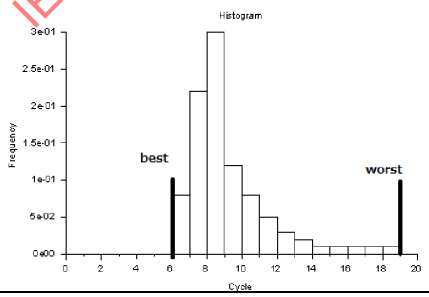


Figure 7—Latency and pitch represent the primary performance characteristics

Table 5—Example of using triplets

	In the simple case in which a single number can describe the specific performance (the triplet has three equal values), the triplet notation is still useful to explicitly note that it has the three equal values. Example: 10.0, 10.0, 10.0
	When the distribution has two numbers, a triplet permits more precise expression. The 'typical' number represents more frequent conditions. Example: 5.0, 5.0, 15.0
	Gaussian distribution may be observed in an ideal case, where the 'typical' value equals the mean. The best and worst represent greater than or equal to three standard deviations away from the mean. Example: 5.1, 10.0, 14.9
	On the real distribution, the measurement of the machine performance limit is sometimes very difficult. In such a case, adopt the 99.9% cumulative point (almost equal to the three standard deviation point on Gaussian distribution) of the distribution as the worst and the 0.1% cumulative point as the best. The typical value represents the mode of the distribution. Example: 6.0, 8.5, 18.9

4.6 Power—PowerConfiguration

Power consumption is a significant concern for many applications. Indeed, some applications must fit in a limited power budget due to thermal limitations, input power limitation or even system stability. Some systems can also be battery-operated, and the power consumption is therefore directly linked to their autonomy.

Modern multi- and many-core platforms often integrate a rich set of heterogeneous components that have very different performance and power profiles (e.g., general-purpose CPUs, DSPs, and GPUs). It is the responsibility of the developers to find the best way to leverage those components to meet the application constraints. To help the developers in their work, many software development tools now integrate some level of power estimation to guide software development. Some tools can even automate software development tasks such as mapping and scheduling based on power information.

Consequently, SHIM 2.0 introduces support for power modeling to provide developers and tools with the appropriate information to perform their work optimally (see Clause 7 for more information). The main classes associated with this feature are listed in Table 6.

To further improve componentization of the SHIM file (see Clause 5), the power-related elements must be defined in a separate file dedicated to power-related information.

Table 6—Component power consumption classes

PowerConfiguration classes	Description
PowerConfiguration	The root element hosting the *PowerConsumptionSet*
PowerConsumptionSet	A *PowerConsumptionSet* contains a list of *PowerConsumption* and *PowerConsumerRef* objects respectively defining operating points, power consumption, and the components whose power consumption is defined.
PowerConsumption	This object defines the power consumption of an operating point.
PowerConsumerRef	A reference to the component whose power consumption is defined by the parent *PowerConsumptionSet*. A *PowerConsumptionSet* can contain one or more *PowerConsumerRef* objects.

The *PowerConfiguration* class is the root element that host the *PowerConsumptionSet* objects defining components power consumption.

The *PowerConsumptionSet* objects allow defining the power consumption of one or more components. Those components are specified by using one or several *PowerConsumerRef* objects, each one of those objects providing a single component reference.

The actual power consumption values are defined using *PowerConsumption* objects, using the *power* and *powerUnit* attributes. As the power consumption of a component depends on the component's operating point, each *PowerConsumption* object also contains a reference to its associated *OperatingPoint*.

4.7 Vendor extensions

SHIM provides a standard interface between the multicore hardware and the software tools. To remain practical and ease its adoption, SHIM follows an evolutionary approach and starts by including the essential elements first, and including new ones as the needs emerge.

However, some end-user might already face those needs and require extending the information provided by SHIM files with additional details. Other users might also need to extend the file with confidential information or with proprietary technology that cannot be included in the standard itself.

To accommodate this issue, SHIM 2.0 provides support for vendor extensions that allow the end-user to integrate SHIM XML files with other files to complement and extend its functionality.

In respect with the goal of increasing the componentization of the SHIM (see Clause 5), all vendor extensions are to be defined in separate files. These files can contain references to one or several SHIM system XML files, as well as other files like, for instance, power consumption files or proprietary technology XML files.

VendorExtension is the root element of any vendor extension file. This element contains one or several objects referencing other files. In SHIM 2.0, the *VendorExtension* element accepts two types of child elements: *SystemConfigurationFile*, which allows referencing a SHIM system XML file, and *PowerConfigurationFile*, which allows referencing a SHIM power XML file. It is allowed to reference several systems and power files.

4.8 Configuration

4.8.1 General

There are two different aspects of configuration in SHIM that are needed by software tools. One aspect is the configuration of software tools based on the basic hardware properties (e.g., cluster organization, number of cores, memory size, processor ISA). These are static hardware properties and tools are able to read the SHIM XML file and configure themselves accordingly. The other aspect is configuring the hardware dynamic properties (e.g., clock frequency, various modes and setting for transfer accelerator) that can be modified according to the system design. For dynamic properties, the tools' user is often required to input the configuration, thus the tools must provide a user interface (either command line or graphical). SHIM provides a mechanism called Common Configuration File (CCF), to serve both for describing the configurable properties and also simultaneously defining the user interface.

Changing the configuration often affects the performance properties. The CCF is designed so that it can also describe how the selection or input value of particular configurable items affect the performance properties.

4.8.2 Common Configuration File (CCF)

The CCF extends SHIM to describe configurable hardware elements and also defines a standard way to generate configuration UI by the tools that support it. The CCF describes the configurable items in a file called CCF XML; this is a separate XML file from the SHIM XML. Software tools using SHIM can utilize this mechanism to provide a configuration tool user interface (see 9.1.3) within its tool, or as a separate standalone tool. When the configuration tool is executed, along with the SHIM XML and CCF, it provides a mechanism to modify the specific parts of SHIM XML, according to the inputs made by the tool user, which can also be automated by the tool.

The SHIM XML and CCF are inter-linked via XPath, the XML Path Language (a query language for selecting nodes from an XML document). In addition, XPath may be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML document.

Figure 8 is an example of an actual CCF.

ccf-sample-for-shim.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ConfigurationSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="CCF Sample for SHIM" xsi:noNamespac
   eSchemaLocation="ccf-schema.xsd">
3    <DefineSet>
4      <Def name="@sclock" path="/SystemConfiguration/ClockFrequency/@clockValue" uri="shim_sample_data.xml"/>
5      <Def name="@cashSize" path="//Cache[@name='UnifiedCache_0_0_0']/@size" uri="shim_sample_data.xml"/>
6    </DefineSet>
7    <Configuration formType="select" name="System clockValue-Select" path="/SystemConfiguration/ClockFrequency/@clockV
   alue" uri="shim_sample_data.xml">
8      <Item key="value" value="20.0"/>
9      <Item key="value" value="40.0"/>
10     <Item key="value" value="100.0"/>
11   </Configuration>
12   <Configuration formType="expression" name="Sample Expression" path="//MasterComponent/ClockFrequency/@clockValue
   " uri="shim_sample_data.xml">
13     <Expression>
14       <description>description</description>
15       <Exp>@sclock * 2</Exp>
16     </Expression>
17   </Configuration>
18   <Configuration formType="text" name="Arch" path="//MasterComponent/@arch" uri="shim_sample_data.xml"/>
19   <Configuration formType="integer" name="nRegister" path="//SharedRegisterCommunication/@nRegister" uri="shim_samp
   le_data.xml"/>
20   <Configuration formType="float" name="ClockFrequency:clockValue" path="/SystemConfiguration/ClockFrequency/@clockV
   alue" uri="shim_sample_data.xml"/>
21   <Configuration formType="bool" name="BooleValue Sample"/>
22 </ConfigurationSet>
23 [EOF]
```

Unicode(UTF-8) 11pt

Figure 8—CCF example

This CCF, when opened by a CCF capable tool, will dynamically create a GUI as shown in Figure 9 (this is a CCF sample application available with source codes from MCA).

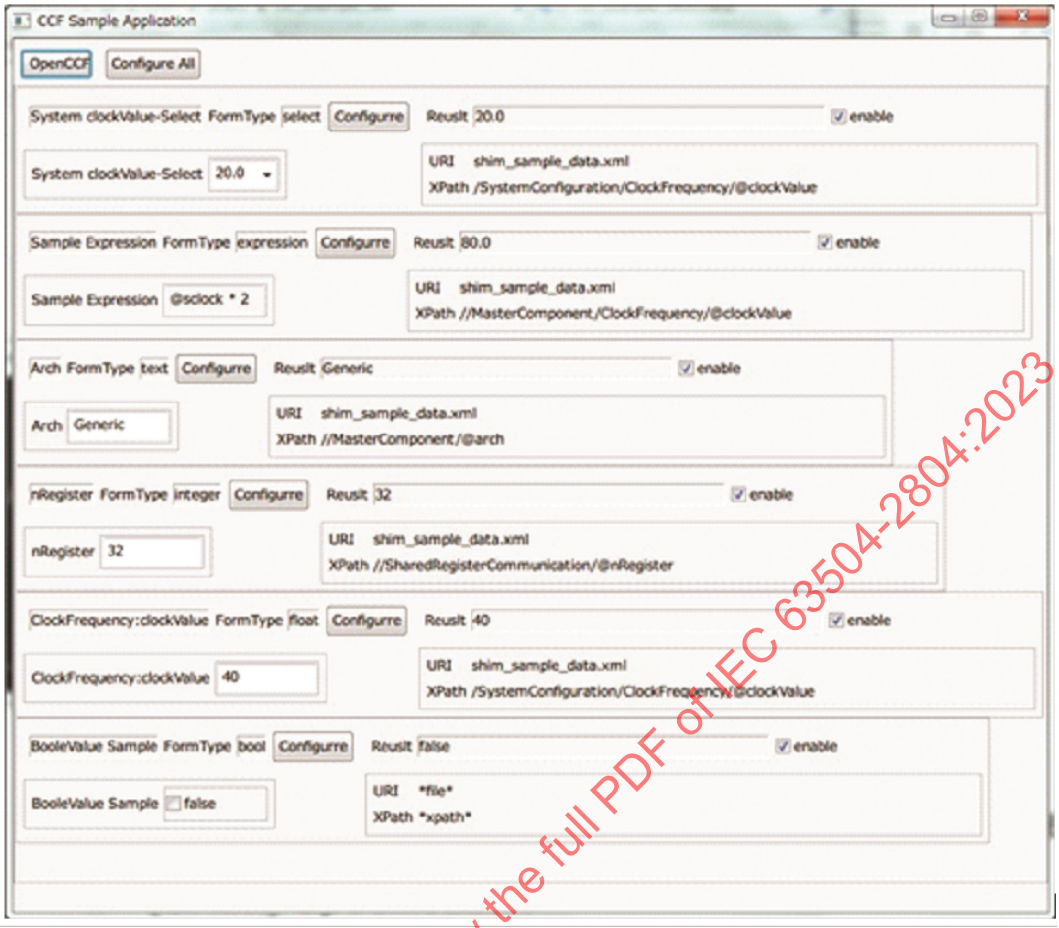


Figure 9—GUI generated by CCF

Please refer to the Clause 9.

5. Roadmap

5.1 General

The first version of SHIM contains the fundamental and critical hardware properties that many tools will find useful. However, as mentioned in 1.4.2, the SHIM authors have decided to follow an evolutionary approach and revise the standard regularly to include new relevant concepts. SHIM is an open technology and wider adoption will fuel its innovative use; this may require enhancements to the specification and the goal is to remain open to such changes.

Properties that are under consideration for future versions of the spec include debugging/trace and basic peripheral components. A few other items are worth mentioning such as componentization of SHIM XML, hardware-related software properties, and schema refinement for smaller XML. Some of those items have already been addressed in SHIM 2.0.

5.2 Further componentization of SHIM XML

SHIM 1.0 only allowed single-file system description and provided no possibility to divide a system description into multiple files. SHIM 2.0 has gone further in the componentization of the system XML file by allowing to define the *CommonInstructionSet* of a processor in a separate XML file. This approach allows avoiding having to repeat identical *Instruction* definitions whenever the same processor core is used several times in a system. It also allows reusing the same XML file across different systems that use the same processor core architecture.

SHIM 2.0 also introduces support for power modeling and vendor extensions with the *PowerConfiguration* and *VendorExtension* classes. To further improve componentization, power-related and vendor extension elements are hosted in separate files.

A remaining challenge of componentizing SHIM XML is that a SHIM XML class, such as *MasterComponent,* contains properties that are static for the hardware board design it is being included with, and also properties that may differ depending on how it is integrated into a particular hardware board. The two must be decoupled in order to reuse the SHIM XML description of the *MasterComponent*. One idea is to use Common Configuration File (CCF), which allows for adjusting the performance value based on some other information, as long as it can be found in the final SHIM XML file or somewhere within the CCF.

Also, along with the componentization of SHIM XML itself, the possibility to align SHIM with IP-XACT where appropriate is being investigated, to ease SHIM XML authoring tools ability to support importing IP-XACT XML files to semi-automate authoring a SHIM XML using the relevant information contained in IP-XACT. The objects contained in the *ComponentSet*, at least the topology part and the object names, should be importable, while others are unique to SHIM XML and must be added.

In future revisions, SHIM could also be aligned with the UPF 3.0 power modeling standard to allow automated or semi-automated extraction of information already encoded in UPF files. For instance, power domain descriptions and operating points could be obtained directly from those files and imported into a platform SHIM model.

5.3 Hardware-related software properties

In addition to the hardware properties that SHIM describes, some tools have a dependency on the system software properties such as the operating system and even some middleware. For example, for a parallelization design aid tool such as a parallelizing compiler, the performance of OS mutual exclusion primitives is critical in deciding on an appropriate lock mechanism for particular processing. Similarly, the tool may need to know the performance of some message passing mechanism. Currently, this kind of information is not included in SHIM, partly because it will require separate SHIM XML files for different system software implementations, along with the library interface definition; a future version of SHIM may extend its coverage into this kind of information.

SHIM 2.0 introduces the power XML files, which allows specifying the power performance of various system components. This information, which is related to hardware, is indeed increasingly more relevant for multicore software as the software implementation of a solution might have to be adapted—possibly automatically by a tool - to lower power consumption and fit the application power budget.

5.4 Schema refinement for smaller XML

The SHIM schema is intended to be simple while allowing it to support both homogeneous and heterogeneous hardware. This has led to using repetitive sets of lines in the XML for homogenous

hardware that have multiple instances of the same component, like a hardware composed of multiple instances of the same cluster configuration. If the clusters are heterogeneous, with each cluster having a different configuration of processing cores, then the number of XML lines does not change but they will have different lines. If SHIM can provide a mechanism to express the redundancy in the schema, the size of SHIM XML file for homogenous hardware can be reduced. This will be considered along with componentization of SHIM XML.

6. SHIM interface

The major part of the SHIM interface is the SHIM XML schema itself. Therefore, understanding the schema comprises the major part of understanding the interface. The basics are described in the Clause 4 and the interface assumes the schema is divided into the following groups:

- Enumeration
- SystemConfiguration
- ComponentSet
- AddressSpaceSet
- CommunicationSet
- FrequencyVoltageSet
- ContentionGroupSet
- PowerConfiguration
- VendorExtension

For each group above, the schema and the description are explained in the following subclauses. For each object or XML element contained in each group, the description and example XML are provided.

The schema is converted into different programming language bindings, using various schema compilers. The SHIM specification does not specify the programming language as this can vary according to the nature of the tools and intended use cases. However, Java is assumed to be one of the primary languages used and the Java class library interface of SHIM, called the SHIM API library, is also provided. Some utility interfaces are defined along with the reference implementation in Java to further ease the programming using the SHIM class libraries.

The following subclauses describe each part, detailing the XML elements and their attributes, along with a pointer to the Java class library interface.

6.1 shim20.xsd

The SHIM XML schema file is as follows. Please refer to the 6.2 through 6.12 for description of elements.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="http://www.multicore-association.org/2017/SHIM2.0/"
           xmlns="http://www.multicore-association.org/2017/SHIM2.0/"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Shim" type="Shim"/>
  <xs:simpleType name="CacheCoherencyType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="HARDWARE"/>
      <xs:enumeration value="SOFTWARE"/>
    </xs:restriction>
```

```
</xs:simpleType>
<xs:simpleType name="CachePrefetchType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="ALWAYS"/>
    <xs:enumeration value="NEVER"/>
    <xs:enumeration value="ONCE"/>
    <xs:enumeration value="ONMISS"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="CacheReplacementType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="FIFO"/>
    <xs:enumeration value="LRU"/>
    <xs:enumeration value="RANDOM"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="CacheType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="DATA"/>
    <xs:enumeration value="INSTRUCTION"/>
    <xs:enumeration value="UNIFIED"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="CacheWriteAllocateType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="ALWAYS"/>
    <xs:enumeration value="NEVER"/>
    <xs:enumeration value="NOFETCH"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="CacheWriteBackType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="ALWAYS"/>
    <xs:enumeration value="NEVER"/>
    <xs:enumeration value="NOFETCH"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="DataRateUnitType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="B/s"/>
    <xs:enumeration value="KiB/s"/>
    <xs:enumeration value="MiB/s"/>
    <xs:enumeration value="GiB/s"/>
    <xs:enumeration value="TiB/s"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="EndianType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="LITTLE"/>
    <xs:enumeration value="BIG"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="FrequencyUnitType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Hz"/>
    <xs:enumeration value="KHz"/>
    <xs:enumeration value="MHz"/>
    <xs:enumeration value="GHz"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="InstructionInputType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="INTEGER"/>
    <xs:enumeration value="FLOAT"/>
    <xs:enumeration value="POINTER"/>
    <xs:enumeration value="IMMEDIATE"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="InstructionOutputType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="INTEGER"/>
    <xs:enumeration value="FLOAT"/>
    <xs:enumeration value="POINTER"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="LockDownType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="NONE"/>
    <xs:enumeration value="LINE"/>
    <xs:enumeration value="WAY"/>
  </xs:restriction>
```

```
</xs:simpleType>
<xs:simpleType name="MasterType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="PU">
      <xs:annotation>
        <xs:documentation>Processing Unit</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="TU">
      <xs:annotation>
        <xs:documentation>Transfer Unit</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="OTHER"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="OperandAddressingType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="NONE"/>
    <xs:enumeration value="IMMEDIATE"/>
    <xs:enumeration value="REGISTER"/>
    <xs:enumeration value="BOTH"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="OperationType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="TAS">
      <xs:annotation>
        <xs:documentation>Test and Set</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="LLSC">
      <xs:annotation>
        <xs:documentation>Load Link/Store Conditional</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="CAX">
      <xs:annotation>
        <xs:documentation>Compare and Exchange</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="OTHER"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="OrderingType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="ORDERED"/>
    <xs:enumeration value="UNORDERED"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="PowerUnitType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="pW"/>
    <xs:enumeration value="nW"/>
    <xs:enumeration value="uW"/>
    <xs:enumeration value="mW"/>
    <xs:enumeration value="W"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="RWType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="RW"/>
    <xs:enumeration value="WX"/>
    <xs:enumeration value="RX"/>
    <xs:enumeration value="R"/>
    <xs:enumeration value="W"/>
    <xs:enumeration value="X"/>
    <xs:enumeration value="RWX"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="SignednessType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="SIGNED"/>
    <xs:enumeration value="UNSIGNED"/>
    <xs:enumeration value="BOTH"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="SizeUnitType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="B"/>
    <xs:enumeration value="KiB"/>
```

```
    <xs:enumeration value="MiB"/>
    <xs:enumeration value="GiB"/>
    <xs:enumeration value="TiB"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ThroughputUnitType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="B/cycle"/>
    <xs:enumeration value="B/Kcycle"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="VoltageUnitType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="pV"/>
    <xs:enumeration value="nV"/>
    <xs:enumeration value="uV"/>
    <xs:enumeration value="mV"/>
    <xs:enumeration value="V"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="AbstractCommunication" abstract="true">
  <xs:sequence>
    <xs:element name="ConnectionSet" type="ConnectionSet"  minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="name" use="required" type="xs:string"/>
</xs:complexType>
<xs:complexType name="AbstractComponent" abstract="true">
    <xs:attribute name="name" use="required" type="xs:string"/>
  <xs:attribute name="id" use="required" type="xs:ID"/>
  <xs:attribute name="voltageDomainRef" use="optional" type="xs:string"/>
  <xs:attribute name="frequencyDomainRef" use="optional" type="xs:string"/>
  <xs:attribute name="operatingPointRef" use="optional" type="xs:string"/>
</xs:complexType>
<xs:complexType name="AbstractInstruction" abstract="true">
  <xs:choice>
    <xs:element name="Performance" type="Performance"  minOccurs="1" maxOccurs="1"/>
  </xs:choice>
  <xs:attribute name="name" use="optional" type="xs:string"/>
  <xs:attribute name="nInputs" use="optional" type="xs:nonNegativeInteger"/>
  <xs:attribute name="nOutputs" use="optional" type="xs:nonNegativeInteger"/>
  <xs:attribute name="SIMDWidth" use="optional" type="xs:nonNegativeInteger"/>
  <xs:attribute name="encodingLength" use="optional" type="xs:nonNegativeInteger"/>
</xs:complexType>
<xs:complexType name="AbstractPerformance" abstract="true">
    <xs:attribute name="best" use="optional" type="xs:float"/>
  <xs:attribute name="typical" use="required" type="xs:float"/>
  <xs:attribute name="worst" use="optional" type="xs:float"/>
</xs:complexType>
<xs:complexType name="Accessor">
  <xs:choice>
    <xs:element name="PerformanceSet" type="PerformanceSet"  minOccurs="0" maxOccurs="unbounded"/>
  </xs:choice>
  <xs:attribute name="masterComponentRef" use="required" type="xs:IDREF"/>
</xs:complexType>
<xs:complexType name="AccessType">
  <xs:sequence minOccurs="1"/>
  <xs:attribute name="name" use="required" type="xs:string"/>
  <xs:attribute name="id" use="required" type="xs:ID"/>
  <xs:attribute name="rwType" use="optional" type="RWType"/>
  <xs:attribute name="accessByteSize" use="optional" type="xs:nonNegativeInteger"/>
  <xs:attribute name="alignmentByteSize" use="optional" type="xs:nonNegativeInteger"/>
  <xs:attribute name="nBurst" use="optional" type="xs:nonNegativeInteger"/>
</xs:complexType>
<xs:complexType name="AccessTypeSet">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="AccessType" type="AccessType"  minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="AddressSpace">
  <xs:choice>
    <xs:element name="SubSpace" type="SubSpace"  minOccurs="0" maxOccurs="unbounded"/>
  </xs:choice>
  <xs:attribute name="name" use="required" type="xs:string"/>
  <xs:attribute name="id" use="required" type="xs:ID"/>
</xs:complexType>
<xs:complexType name="AddressSpaceSet">
  <xs:choice>
    <xs:element name="AddressSpace" type="AddressSpace"  minOccurs="1" maxOccurs="unbounded"/>
  </xs:choice>
</xs:complexType>
<xs:complexType name="Cache">
```

```
    <xs:complexContent>
      <xs:extension base="AbstractComponent">
        <xs:sequence>
          <xs:element name="CacheRef" type="xs:IDREF" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="cacheType" use="required" type="CacheType">
          <xs:annotation>
            <xs:documentation>soft / hard</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="cacheCoherency" use="required" type="CacheCoherencyType"/>
        <xs:attribute name="size" use="required" type="xs:nonNegativeInteger"/>
        <xs:attribute name="sizeUnit" use="required" type="SizeUnitType"/>
        <xs:attribute name="nWay" use="optional" default="1" type="xs:nonNegativeInteger"/>
        <xs:attribute name="lineSize" use="optional" default="16" type="xs:nonNegativeInteger"/>
        <xs:attribute name="lockDownType" use="optional" type="LockDownType"/>
        <xs:attribute name="prefetch" use="optional" type="CachePrefetchType"/>
        <xs:attribute name="replacement" use="optional" type="CacheReplacementType"/>
        <xs:attribute name="prefetchDistance" use="optional" type="xs:nonNegativeInteger"/>
        <xs:attribute name="writeAllocate" use="optional" type="CacheWriteAllocateType"/>
        <xs:attribute name="writeBack" use="optional" type="CacheWriteBackType"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="CommonInstructionSet">
    <xs:choice>
      <xs:element name="FunctionalUnitSet" type="FunctionalUnitSet"  minOccurs="0" maxOccurs="1"/>
      <xs:element name="FunctionalUnitSetFile" type="FunctionalUnitSetFile"  minOccurs="0"
maxOccurs="1"/>
    </xs:choice>
    <xs:attribute name="name" use="required" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="CommunicationSet">
    <xs:choice>
      <xs:element name="SharedRegisterCommunication" type="SharedRegisterCommunication"  minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="SharedMemoryCommunication" type="SharedMemoryCommunication"  minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="EventCommunication" type="EventCommunication"  minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="FIFOCommunication" type="FIFOCommunication"  minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="InterruptCommunication" type="InterruptCommunication"  minOccurs="0"
maxOccurs="unbounded"/>
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="ComponentSet">
    <xs:complexContent>
      <xs:extension base="AbstractComponent">
        <xs:sequence>
          <xs:element name="ComponentSet" type="ComponentSet"  minOccurs="0" maxOccurs="unbounded"/>
          <xs:element name="MasterComponent" type="MasterComponent"  minOccurs="0"
maxOccurs="unbounded"/>
          <xs:element name="SlaveComponent" type="SlaveComponent"  minOccurs="0"
maxOccurs="unbounded"/>
          <xs:element name="Cache" type="Cache"  minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Connection">
    <xs:choice>
      <xs:element name="Performance" type="Performance"  minOccurs="0" maxOccurs="unbounded"/>
    </xs:choice>
    <xs:attribute name="from" use="required" type="xs:IDREF">
      <xs:annotation>
        <xs:documentation>Reference to the instance of MasterComponent</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="to" use="required" type="xs:IDREF">
      <xs:annotation>
        <xs:documentation>Reference to the instance of MasterComponent</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="ConnectionSet">
    <xs:choice>
      <xs:element name="Connection" type="Connection"  minOccurs="1" maxOccurs="unbounded"/>
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="ContentionGroup">
```

```
    <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="1">
        <xs:element name="Throughput" type="Throughput"/>
        <xs:element name="DataRate" type="DataRate"/>
      </xs:choice>
      <xs:element name="PerformanceSetRef" type="xs:IDREF" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" use="required" type="xs:ID"/>
    <xs:attribute name="name" use="required" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="ContentionGroupSet">
    <xs:sequence>
      <xs:element name="ContentionGroup" type="ContentionGroup"  minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="CustomInstruction">
    <xs:complexContent>
      <xs:extension base="AbstractInstruction">
        <xs:sequence>
          <xs:element name="InstructionInput" type="InstructionInput"  minOccurs="0"
maxOccurs="unbounded"/>
          <xs:element name="InstructionOperation" type="InstructionOperation"  minOccurs="1"
maxOccurs="unbounded"/>
          <xs:element name="InstructionOutput" type="InstructionOutput"  minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="DataRate">
      <xs:attribute name="value" use="required" type="xs:nonNegativeInteger"/>
    <xs:attribute name="unit" use="optional" type="DataRateUnitType"/>
  </xs:complexType>
  <xs:complexType name="EventCommunication">
    <xs:complexContent>
      <xs:extension base="AbstractCommunication">
            </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="FIFOCommunication">
    <xs:complexContent>
      <xs:extension base="AbstractCommunication">
              <xs:attribute name="dataSize" use="required" type="xs:nonNegativeInteger"/>
        <xs:attribute name="dataSizeUnit" use="optional" type="SizeUnitType"/>
        <xs:attribute name="queueSize" use="required" type="xs:nonNegativeInteger"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="FrequencyDomain">
      <xs:attribute name="name" use="required" type="xs:string"/>
    <xs:attribute name="id" use="required" type="xs:ID"/>
  </xs:complexType>
  <xs:complexType name="FrequencyVoltageSet">
    <xs:sequence>
      <xs:element name="FrequencyDomain" type="FrequencyDomain"  minOccurs="1" maxOccurs="unbounded"/>
      <xs:element name="VoltageDomain" type="VoltageDomain"  minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="OperatingPointSet" type="OperatingPointSet"  minOccurs="1"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="FunctionalUnit">
    <xs:sequence>
      <xs:element name="Instruction" type="Instruction"  minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="CustomInstruction" type="CustomInstruction"  minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" use="required" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="FunctionalUnitSet">
    <xs:choice>
      <xs:element name="FunctionalUnit" type="FunctionalUnit"  minOccurs="1" maxOccurs="unbounded"/>
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="FunctionalUnitSetFile">
      <xs:attribute name="src" use="required" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="Instruction">
    <xs:complexContent>
      <xs:extension base="AbstractInstruction">
              <xs:attribute name="operation" use="required" type="xs:string"/>
        <xs:attribute name="inputBitWidth" use="required" type="xs:nonNegativeInteger"/>
```

```
        <xs:attribute name="outputBitWidth" use="required" type="xs:nonNegativeInteger"/>
        <xs:attribute name="supportedSignedness" use="optional" type="SignednessType"/>
        <xs:attribute name="operandAddressing" use="optional" type="OperandAddressingType"/>
        <xs:attribute name="immediateBitWidth" use="optional" type="xs:nonNegativeInteger"/>
        <xs:attribute name="isEmulated" use="optional" type="xs:boolean"/>
        <xs:attribute name="inputPreserved" use="optional" type="xs:boolean"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="InstructionInput">
      <xs:attribute name="id" use="required" type="xs:ID"/>
    <xs:attribute name="bitWidth" use="required" type="xs:nonNegativeInteger"/>
    <xs:attribute name="type" use="required" type="InstructionInputType"/>
    <xs:attribute name="value" use="optional" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="InstructionOperation">
    <xs:sequence>
      <xs:element name="InstructionOperand" type="xs:IDREF" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="operation" use="optional" type="xs:string"/>
    <xs:attribute name="id" use="optional" type="xs:ID"/>
  </xs:complexType>
  <xs:complexType name="InstructionOutput">
      <xs:attribute name="bitWidth" use="optional" type="xs:nonNegativeInteger"/>
    <xs:attribute name="type" use="optional" type="InstructionOutputType"/>
    <xs:attribute name="ref" use="optional" type="xs:IDREF"/>
  </xs:complexType>
  <xs:complexType name="InterruptCommunication">
    <xs:complexContent>
      <xs:extension base="AbstractCommunication">
            </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Latency">
    <xs:complexContent>
      <xs:extension base="AbstractPerformance">
            </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="MasterComponent">
    <xs:complexContent>
      <xs:extension base="AbstractComponent">
        <xs:sequence>
          <xs:element name="CommonInstructionSet" type="CommonInstructionSet"  minOccurs="0"
maxOccurs="1"/>
          <xs:element name="CacheRef" type="xs:IDREF" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element name="AccessTypeSet" type="AccessTypeSet"  minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
        <xs:attribute name="masterType" use="required" type="MasterType"/>
        <xs:attribute name="arch" use="required" type="xs:string"/>
        <xs:attribute name="archOption" use="optional" type="xs:string"/>
        <xs:attribute name="nChannel" use="optional" type="xs:nonNegativeInteger"/>
        <xs:attribute name="pid" use="optional" type="xs:string"/>
        <xs:attribute name="endian" use="optional" default="LITTLE" type="EndianType"/>
        <xs:attribute name="nThread" use="optional" type="xs:nonNegativeInteger"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="MasterSlaveBinding">
    <xs:choice>
      <xs:element name="Accessor" type="Accessor"  minOccurs="1" maxOccurs="unbounded"/>
    </xs:choice>
    <xs:attribute name="slaveComponentRef" use="required" type="xs:IDREF"/>
  </xs:complexType>
  <xs:complexType name="MasterSlaveBindingSet">
    <xs:choice>
      <xs:element name="MasterSlaveBinding" type="MasterSlaveBinding"  minOccurs="1"
maxOccurs="unbounded"/>
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="MemoryConsistencyModel">
      <xs:attribute name="rawOrdering" use="optional" type="OrderingType">
      <xs:annotation>
        <xs:documentation>Read After Write</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="warOrdering" use="optional" type="OrderingType">
      <xs:annotation>
        <xs:documentation>Write After Read</xs:documentation>
      </xs:annotation>
    </xs:attribute>
```

```
    <xs:attribute name="wawOrdering" use="optional" type="OrderingType">
      <xs:annotation>
        <xs:documentation>Write After Write</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="rarOrdering" use="optional" type="OrderingType"/>
  </xs:complexType>
  <xs:complexType name="OperatingPoint">
      <xs:attribute name="name" use="required" type="xs:string"/>
    <xs:attribute name="id" use="optional" type="xs:ID"/>
    <xs:attribute name="frequency" use="required" type="xs:nonNegativeInteger"/>
    <xs:attribute name="frequencyUnit" use="optional" type="FrequencyUnitType"/>
    <xs:attribute name="voltage" use="optional" type="xs:nonNegativeInteger"/>
    <xs:attribute name="voltageUnit" use="optional" type="VoltageUnitType"/>
  </xs:complexType>
  <xs:complexType name="OperatingPointSet">
    <xs:choice>
      <xs:element name="OperatingPoint" type="OperatingPoint"  minOccurs="1" maxOccurs="unbounded"/>
    </xs:choice>
    <xs:attribute name="name" use="required" type="xs:string"/>
    <xs:attribute name="id" use="required" type="xs:ID"/>
  </xs:complexType>
  <xs:complexType name="Performance">
    <xs:sequence>
      <xs:element name="Pitch" type="Pitch"  minOccurs="1" maxOccurs="1"/>
      <xs:element name="Latency" type="Latency"  minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="accessTypeRef" use="optional" type="xs:IDREF"/>
  </xs:complexType>
  <xs:complexType name="PerformanceSet">
    <xs:choice>
      <xs:element name="Performance" type="Performance"  minOccurs="0" maxOccurs="unbounded"/>
    </xs:choice>
    <xs:attribute name="id" use="optional" type="xs:ID"/>
    <xs:attribute name="cacheRef" use="optional" type="xs:IDREF"/>
  </xs:complexType>
  <xs:complexType name="Pitch">
    <xs:complexContent>
      <xs:extension base="AbstractPerformance">
            </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="PowerConfiguration">
    <xs:sequence>
      <xs:element name="PowerConsumptionSet" type="PowerConsumptionSet"  minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="systemConfigurationSrc" use="optional" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="PowerConfigurationFile">
      <xs:attribute name="id" use="optional" type="xs:ID"/>
    <xs:attribute name="name" use="optional" type="xs:string"/>
    <xs:attribute name="src" use="optional" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="PowerConsumption">
      <xs:attribute name="operatingPointRef" use="required" type="xs:string"/>
    <xs:attribute name="power" use="optional" type="xs:nonNegativeInteger"/>
    <xs:attribute name="powerUnit" use="optional" type="PowerUnitType"/>
  </xs:complexType>
  <xs:complexType name="PowerConsumptionSet">
    <xs:sequence>
      <xs:element name="PowerConsumerRef" type="xs:string" minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>Reference to ComponentSet, SlaveComponent, MasterComponent, Cache, or
FunctionalUnit</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="PowerConsumption" type="PowerConsumption"  minOccurs="1"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" use="optional" type="xs:ID"/>
    <xs:attribute name="name" use="optional" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="SharedMemoryCommunication">
    <xs:complexContent>
      <xs:extension base="AbstractCommunication">
              <xs:attribute name="operationType" use="optional" type="OperationType"/>
        <xs:attribute name="dataSize" use="optional" type="xs:nonNegativeInteger"/>
        <xs:attribute name="dataSizeUnit" use="optional" type="SizeUnitType"/>
        <xs:attribute name="addressSpaceRef" use="optional" type="xs:IDREF"/>
        <xs:attribute name="subSpaceRef" use="optional" type="xs:IDREF"/>
```

```
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="SharedRegisterCommunication">
    <xs:complexContent>
      <xs:extension base="AbstractCommunication">
              <xs:attribute name="dataSize" use="required" type="xs:nonNegativeInteger"/>
        <xs:attribute name="dataSizeUnit" use="required" type="SizeUnitType"/>
        <xs:attribute name="nRegister" use="required" type="xs:nonNegativeInteger"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Shim">
    <xs:choice>
      <xs:element name="SystemConfiguration" type="SystemConfiguration"  minOccurs="1" maxOccurs="1"/>
      <xs:element name="PowerConfiguration" type="PowerConfiguration"  minOccurs="1" maxOccurs="1"/>
      <xs:element name="VendorExtension" type="VendorExtension"  minOccurs="1" maxOccurs="1"/>
      <xs:element name="FunctionalUnitSet" type="FunctionalUnitSet"  minOccurs="1" maxOccurs="1"/>
    </xs:choice>
    <xs:attribute name="name" use="required" type="xs:string"/>
    <xs:attribute name="shimVersion" use="required" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="SlaveComponent">
    <xs:annotation>
      <xs:documentation>Memory</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="AbstractComponent">
              <xs:attribute name="size" use="required" type="xs:nonNegativeInteger"/>
        <xs:attribute name="sizeUnit" use="required" type="SizeUnitType"/>
        <xs:attribute name="rwType" use="required" type="RWType"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="SubSpace">
    <xs:choice>
      <xs:element name="MemoryConsistencyModel" type="MemoryConsistencyModel"  minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="MasterSlaveBindingSet" type="MasterSlaveBindingSet"  minOccurs="0"
maxOccurs="1"/>
    </xs:choice>
    <xs:attribute name="name" use="required" type="xs:string"/>
    <xs:attribute name="id" use="required" type="xs:ID"/>
    <xs:attribute name="start" use="required" type="xs:long"/>
    <xs:attribute name="end" use="required" type="xs:long"/>
    <xs:attribute name="endian" use="optional" type="EndianType"/>
  </xs:complexType>
  <xs:complexType name="SystemConfiguration">
    <xs:sequence>
      <xs:element name="ComponentSet" type="ComponentSet"  minOccurs="1" maxOccurs="1"/>
      <xs:element name="CommunicationSet" type="CommunicationSet"  minOccurs="0" maxOccurs="1"/>
      <xs:element name="AddressSpaceSet" type="AddressSpaceSet"  minOccurs="0" maxOccurs="1"/>
      <xs:element name="FrequencyVoltageSet" type="FrequencyVoltageSet"  minOccurs="1" maxOccurs="1"/>
      <xs:element name="ContentionGroupSet" type="ContentionGroupSet"  minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="SystemConfigurationFile">
      <xs:attribute name="id" use="optional" type="xs:ID"/>
    <xs:attribute name="name" use="optional" type="xs:string"/>
    <xs:attribute name="src" use="optional" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="Throughput">
      <xs:attribute name="value" use="required" type="xs:nonNegativeInteger"/>
    <xs:attribute name="unit" use="optional" type="ThroughputUnitType"/>
    <xs:attribute name="frequencyDomainRef" use="required" type="xs:IDREF"/>
  </xs:complexType>
  <xs:complexType name="VendorExtension">
    <xs:sequence>
      <xs:element name="SystemConfigurationFile" type="SystemConfigurationFile"  minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="PowerConfigurationFile" type="PowerConfigurationFile"  minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="VoltageDomain">
      <xs:attribute name="name" use="required" type="xs:string"/>
    <xs:attribute name="id" use="required" type="xs:ID"/>
  </xs:complexType>
</xs:schema>
```

6.2 Conventions

The following shows the conventions of the SHIM interfaces:

— The interface is grouped into Enumeration, SystemConfiguration, ComponentSet, AddressSpaceSet, CommunicationSet, FrequencyVoltageSet, ContentionGroupSet, PowerConfiguration, and VendorExtension.

— Each group has a **Schema** and **Description**.

— Each group describes its objects in separate subclauses. These have a **Description** and **Example**.

— The objects and attributes use **bold** style, and the types use italic.

6.3 Enumeration

Schema

Figure 10 is an example of a schema of Enumeration:

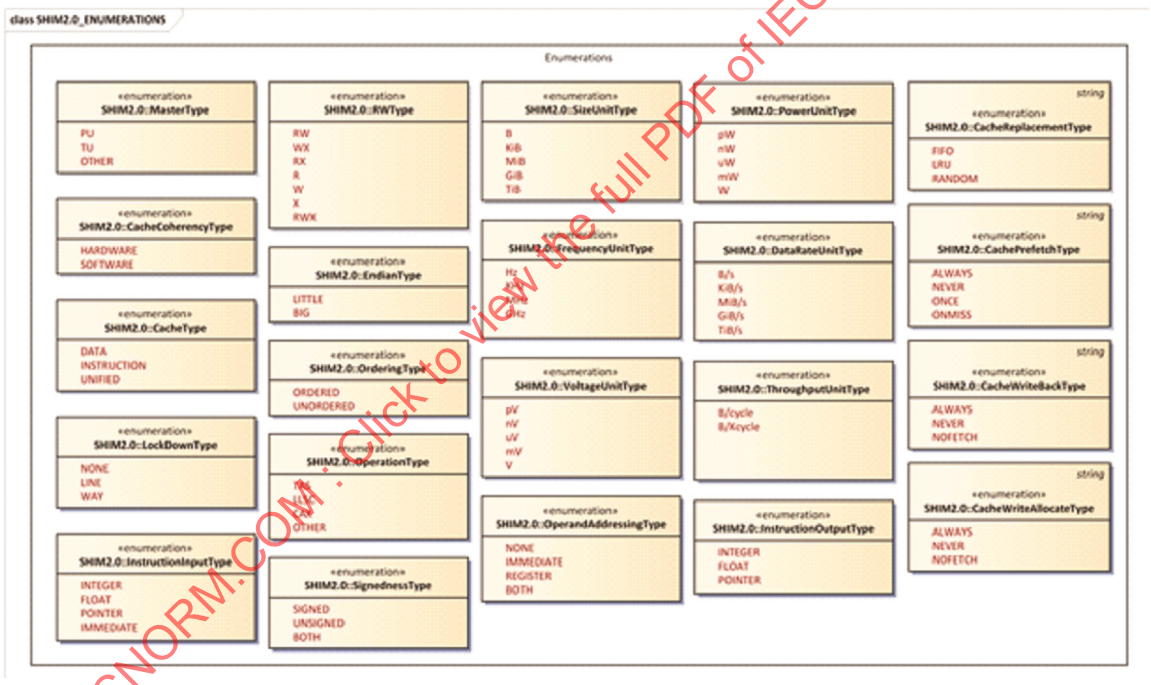


Figure 10—Schema

Description

An enumeration is a special group that defines various constants used in some of the SHIM object attributes. The objects use the constants as values for selected attributes. When the attributes take one enumeration as its value, its attribute types specify which enumeration type it uses.

The following enumeration types are defined:

— ***MasterType*** specifies a type of *MasterComponent*. The values can be one of PU (Processor Unit, such as CPU), TU (Transfer Unit, such as DMA), or OTHER.

— ***CacheCoherencyType*** specifies the type of cache coherency mechanism supported. It can be either HARDWARE for hardware-based coherency or SOFTWARE for software-based coherency.

— ***CacheType*** specifies the type of cache, which can be DATA for data cache, INSTRUCTION for the instruction cache, and UNIFIED for a unified cache.

— ***LockDownType*** specifies the type of supported cache content lockdown operation. The values can be one of LINE for line-lockdown, WAY for way-lockdown, and NONE if the lockdown is not supported.

— ***RWType*** specifies memory access types, which can be R for reading, W for writing, X for executing, RW for both R and W, RWX for all of R, W and X, WX for both W and X, and RX for both R and X.

— ***EndianType*** specifies the endian, or byte-order.

— ***OrderingType*** specifies the memory consistency model. It can be ORDERED for ordered memory consistency or UNORDERED for unordered memory consistency.

— ***OperandAddressingType*** specifies the type of operands, which can be NONE, IMMEDIATE for an immediate encoded in the instruction, REGISTER for a register reference, or BOTH.

— ***OperationType*** specifies the type of shared memory communication, which can be TAS for Test and Set, LLSC for Load-link/Store Conditional, CAX for Compare and Exchange, and OTHER for other unspecified operation.

— ***SizeUnitType*** specifies the unit for data size, which can be B for byte, KiB for kilo binary byte, MiB for mega binary byte, GiB for Giga binary byte, and TiB for Tera binary byte.

— ***FrequencyUnitType*** specifies the unit for frequencies, which can be Hz, kHz, MHz, or GHz.

— ***VoltageUnitType*** specifies the unit for voltages, which can be V, mV, uV, nV, or pV.

— ***PowerUnitType*** specifies the unit for power values, which can be W, mW, uW, nW, or pW.

— ***DataRateUnitType*** specifies the unit for data rate, in terms of amount of data transferred per second, which can be B/s for byte per second, KiB/s for kilo binary byte per second, MiB/s for mega binary byte per second, GiB/s for Giga binary byte per second or TiB/s for tera binary byte per second.

— ***ThroughputUnitType*** specifies the unit for data throughput, in term of amount of data transferred per cycle, which can be B/cycle for byte per cycle, B/Kcycle for byte per kilo cycle (1000 cycles).

— ***SignednessType*** specifies the supported signedness of an instruction, which could be SIGNED if the instruction exclusively supports signed operation (e.g., signed multiply), UNSIGNED is the instruction exclusively supports unsigned operations (e.g., unsigned multiply), or BOTH if the instruction supports both types of operation.

— ***InstructionInputType*** specifies the type of an instruction input, the input type can either be INTEGER if it is an integer value, FLOAT it is a float or double value, POINTER if it is a memory pointer, or IMMEDIATE if it is a constant value included in the instruction binary code.

— ***InstructionOutputType*** specifies the type of an instruction output, the output type can either be INTEGER if it is an integer value, FLOAT it is a float or double value, or a POINTER if it is a memory pointer.

— ***CacheReplacementType*** specifies the replacement policy of cache. The policy can be FIFO (first-in, first-out), LRU (least recently used) or random

— ***CachePrefetchType*** specifies the prefetch mechanism of a cache. The mechanism can be ALWAYS if the cache always initiates a prefetch, ONMISS, if it prefetches only on the cache misses, ONCE if it prefetches only on the first cache miss per cache line, or NEVER if it never prefetches.

— ***CacheWriteBackType*** specifies the write-back policy of a cache. The policy can be ALWAYS if the dirty data is always held in the cache and written back towards memory later, NEVER if the cache implements a write-through policy, or NOFETCH if the dirty data is held in the cache as long as no fetch is required.

NOTE—A fetch would be required if the write was not for an integral number of cache lines.

— ***CacheWriteAllocateType*** specifies the write allocation policy of a cache. The policy can be ALWAYS if the cache allocates a new cache line on every write miss, NEVER if it never allocates a new cache line, or NOFETCH if it allocates a new cache line on a write miss as long as no fetch is required.

NOTE—A fetch would be required if the write was not for an integral number of cache lines.

Example

See examples for objects that use these types in the following subclauses.

6.4 Shim

The *Shim* object is a root object. All shim XML starts with this object as root.

It may have only one of the following objects: *SystemConfiguration*, *PowerConfiguration*, *VendorExtension*, or *FunctionalUnitSet*.

— **name** (mandatory; type: *string*): The name of this SHIM description.

— **shimVersion** (mandatory; type: *string*): The version SHIM interface specification. For this version of SHIM interface, it is “2.0”. It may be trailed with minor revision numbers (e.g., “2.0.1”).

Example

```
<shim:Shim
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:shim="http://www.multicore-association.org/2017/SHIM2.0/"
    xsi:schemaLocation="http://www.multicore-association.org/2017/SHIM2.0/ shim20.xsd"
    name="MySystem" shimVersion="2.0">
<SystemConfiguration>
    <ComponentSet name="Cluster_0"></ComponentSet>
    <CommunicationSet></CommunicationSet>
    <AddressSpaceSet></AddressSpaceSet>
    <FrequencyVoltageSet>
        <FrequencyDomain name="fd_main"/>
        <OperatingPointSet>
            <OperatingPoint name="op_default" frequency="500" frequencyUnit="MHz"/>
        </OperatingPointSet>
    </FrequencyVoltageSet>
</SystemConfiguration>
</shim:Shim>
```

6.5 SystemConfiguration

Schema

Figure 11 is an example of a schema of SystemConfiguration.

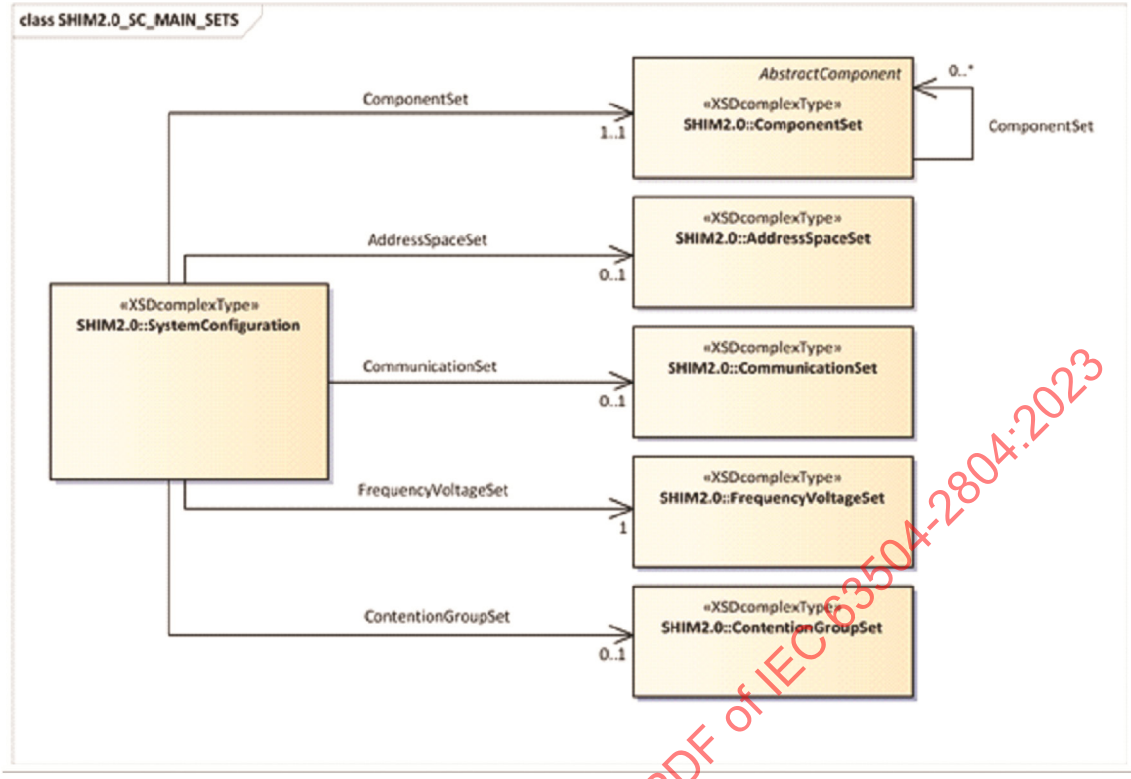


Figure 11—Schema

Description

All SHIM system description XML starts with the *SystemConfiguration* object.

A *SystemConfiguration* object has one *ComponentSet*, one *FrequencyVoltageSet*, and zero or one *AddressSpaceSet*, *CommunicationSet*. and *ContentionGroupSet*.

Refer to *ComponentSet*, *FrequencyVoltageSet,* *AddressSpaceSet*, *CommunicationSet* and *ContentionGroupSet* for more information about those objects.

Example

See Shim (6.4).

6.6 ComponentSet

Schema

Figure 12 is an example of a schema of ComponentSet.

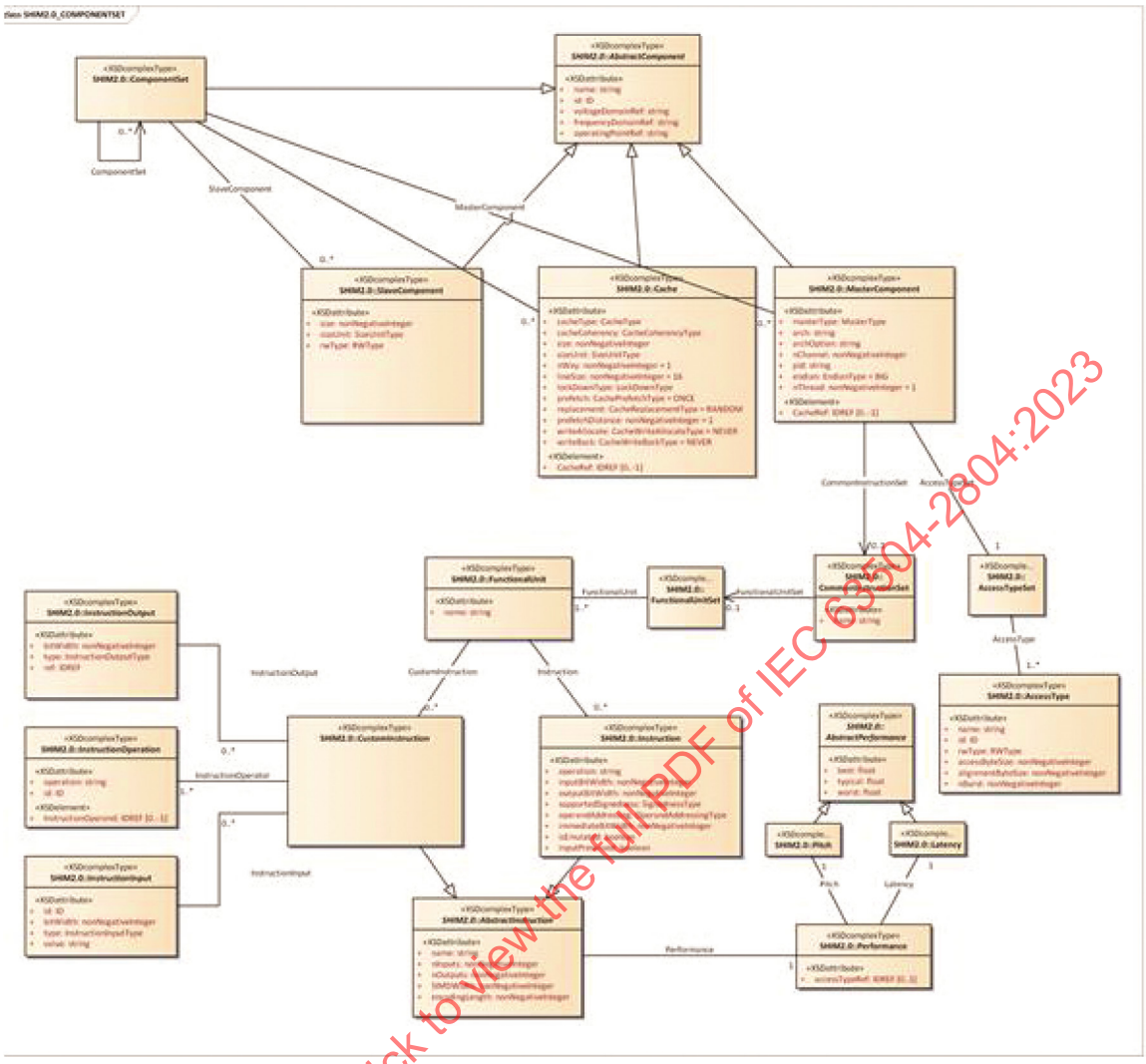


Figure 12—Schema

Description

ComponentSet is the root of the hardware component topology description that SHIM contains. It has **a name** (mandatory) attribute. It may have *MasterComponent*, Cache, *SlaveComponent*, or another *ComponentSet*.

6.6.1 MasterComponent

Description

MasterComponent is a processor core, accelerator (including DMA accelerator), or any other type of component that can be a master. It has the following objects and/or attributes.

— **AccessTypeSet** (mandatory): Refer to AccessTypeSet (6.6.4).

— **CommonInstructionSet** (optional): Refer to CommonInstructionSet (6.6.6).

— **name** (mandatory; type *string*): The name of this object. It should follow the same text used in the hardware reference manual.

— **id** (mandatory; type *ID*): The ID of this object.

— **masterType** (mandatory): The type of master and ***MasterType.***

— **arch** (mandatory; type *string*): Specifies the name of this component's architecture and is intended mostly for describing processor instruction architecture. It is advised to use the official identifier for the ISA generally found in the architecture reference manual or similar.

— **archOption** (optional): Specifies additional architecture properties.

— **pid** (optional): The ID of this *MasterComponent*. It is intended to be used for processor core id when the processor has some way of identifying the processor core when there are multiple cores. The scheme for describing the processor ID can be different, and it should follow the semantics used in the architecture reference manual of the processor. Since ID may not be expressed by an integer or single integer, this attribute is of type *string*.

— **nChannel** (optional; type *nonNegativeInteger*): Specifies the number of channels and is intended for describing a number of channels of DMA, when **masterType** is **TU**.

— **nThread** (optional; type *nonNegativeInteger*): Specifies the number of hardware thread, and intended for a processor core that supports hardware-threading.

— **endian** (optional; type *EndianType*): The endianness of this object.

— **frequencyDomainRef** (optional; type *IDREF*): The ID of the related frequency domain, see FrequencyDomain (6.7.1) for more information.

— **voltageDomainRef** (optional; type *IDREF*): The ID of the related voltage domain, see VoltageDomain (6.7.2) for more information

— **operatingPointSetRef** (optional; type *ID*): The ID of the related operating point set, see OperatingPointSet (6.7.3) for more information.

— **CacheRef** (optional; type *IDREF*): Specifies the *id* of a *Cache* object that is one level from *MasterComponent*. Several CacheRef elements can be specified.

Example

```
<MasterComponent name="Core_0_0_0" id="SHIMEDITOR25331849408820141005130142974"
    masterType="PU" arch="Generic" archOption="" pid="16" nThread="1" endian="LITTLE">
    <CommonInstructionSet name="LLVM Instructions">
    <FunctionalUnit name="B">
        <Instruction name="B_IMM" operation="br" nInputs="1" nOutputs="1"
outputBitWidth="64"
                encodingLength="32" addressingMode="IMMEDIATE"
addressingImmWidth="64">
            <Performance>
                <Pitch   best="1.0" typical="1.0" worst="1.0" />
                <Latency best="1.0" typical="1.0" worst="1.0" />
            </Performance>
        </Instruction>
    <CacheRef> SHIMEDITOR86224119018201410051301455548</CacheRef>
    </CommonInstructionSet>


    <AccessTypeSet>
        <AccessType name="AT_0_0_0_0" id="SHIMEDITOR272374223931201410051301455551"
    rwType="R" accessByteSize="4" alignmentByteSize="4" nBurst="8"/>
    ...
    </AccessTypeSet>
</MasterComponent>
```

6.6.2 SlaveComponent

Description

SlaveComponent is for describing a slave device such as memory. It has the following objects and/or attributes:

- **name** (mandatory; type *string*): The name of this object.
- **id** (mandatory; type *ID*): The ID of this object.
- **size** (mandatory; type *nonNegativeInteger*): The size of this memory.
- **sizeUnit** (mandatory; type *SizeUnitType*): The unit of **size**.
- **rwType** (mandatory; type *RWType*): Specifies this memory is readable and/or writable.
- **frequencyDomainRef** (optional; type *ID*): The ID of the related frequency domain, see FrequencyDomain (6.7.1) for more information.
- **voltageDomainRef** (optional; type *ID*): The ID of the related voltage domain, see VoltageDomain (6.7.2) for more information
- **operatingPointSetRef** (optional; type *ID*): The ID of the related operating point set, see OperatingPointSet (6.7.3) for more information.

Example

```
<SlaveComponent name="Memory_0_0_0" id="SHIMEDITOR81817748650201410051301429 75" size="128"
    sizeUnit="KiB" rwType="RW"/>
```

6.6.3 Cache

Description

This object describes a cache with the following objects and/or attributes:

- **name** (mandatory; type *string*): The name of this object.
- **id** (mandatory; type *ID*): The ID of this object.
- **cacheType** (mandatory; type *CacheType*): Specifies this cache type.
- **cacheCoherency** (mandatory; type *CacheCoherencyType*): Specifies what cache coherency mechanism is provided.
- **size** (mandatory; type *nonNegativeInteger*): This cache size.
- **sizeUnit (**mandatory; type *SizeUnitType)*: The unit of **size**.
- **nWay** (optional; type *nonNegativeInteger*): Specifies the number of cache ways.
- **lineSize** (optional; type *nonNegativeInteger*): Specifies the cache line size.
- **lockDownType** (optional; type *LockDownType*): Specifies the supported cache lock down operation.
- **prefetch** (optional; type *CachePrefetchType*): Defines the cache prefetch policy.
- **replacement** (optional; type *CacheReplacementType*): Defines the cache replacement policy.

— **prefetchDistance** (optional; type *nonNegativeInteger*): Defines the number of cache lines that are prefetched (for the selected prefetch policy).

— **writeAllocate** (optional; type *CacheWriteAllocateType*): Defines the cache write allocation policy.

— **writeBack** (optional; type *CacheWriteBackType*): Defines the cache writeback policy.

— **CacheRef** (optional; type *IDREF*): Specifies the *id* of another *Cache* that is one level away from *MasterComponent*. Several CacheRef elements can be specified.

— **frequencyDomainRef** (optional; type *ID*): The ID of the related frequency domain, see FrequencyDomain for more information.

— **voltageDomainRef** (optional; type *ID*): The ID of the related voltage domain, VoltageDomain for more information.

— **operatingPointSetRef** (optional; type *ID*): The ID of the OperatingPointSet defining the operating points of this component.

Example

```
<Cache name="UnifiedCache_0_0_0" id="SHIMEDITOR86224119018201410051301455548"
  cacheType="UNIFIED" cacheCoherency="SOFTWARE" size="64" sizeUnit="KiB" nWay="16"
  lineSize="128" lockDownType="LINE">
  <CacheRef>SHIMEDITOR862241190182014100513014554 9</CacheRef>
</Cache>
```

6.6.4 AccessTypeSet

Description

This object bundles one or more AccessType. It has the following objects and/or attributes:

— **AccessType** (mandatory): Refer to AccessType (6.6.5).

Example

```
<AccessTypeSet>
    <AccessType name="AT_0_0_0_0" id="SHIMEDITOR27237422393120141005130145551"
rwType="R"
                accessByteSize="4" alignmentByteSize="4" nBurst="8"/>
    <AccessType name="AT_0_0_0_1" id="SHIMEDITOR29805129821320141005130145552"
rwType="R"
                accessByteSize="8" alignmentByteSize="8" nBurst="8"/>
</AccessTypeSet>
```

6.6.5 AccessType

Description

This object describes the type of access, mostly intended for, but not limited to, memory access by a processor. It has the following objects and/or attributes.

— **name** (mandatory; type *string*): The name of this object.

— **id** (mandatory; type *ID*): The ID of this object.

— **rwType** (optional; type *RWType*): Specifies the type of access.

— **accessByteSize** (optional; type *nonNegativeInteger*): Specifies the data size of access in bytes.

— **alignmentByteSize** (optional; type *nonNegativeInteger*): Specifies the alignment requirement in byte of this access.

— **nBurst** (optional; type *nonNegativeInteger*): Specifies the burst length. The burst size is accessByteSize. It is mostly intended for masterType=TU.

Example

See AccessTypeSet (6.6.4).

6.6.6 CommonInstructionSet

Description

This object allows specifying the *Instructions* and *CustomInstructions* supported by the *MasterComponent* and the *FunctionalUnits* that composes it.

The *MasterComponent*'s Instructions and *CustomInstructions* are defined in terms of instructions or sequence of instructions of a reference instruction set [see Instruction (6.6.10) and CustomInstruction (6.6.11) for more details]. This reference instruction set is expected to be the LLVM instruction set,[6] although it is not explicitly enforced by the XML schema to allow for extensions or alternative instruction sets.

The *CommonInstructionSet* can either contain a *FunctionalUnitSet* or a *FunctionalUnitSetFile*, but not both.

a) **name** (mandatory; type *string*): The name of the supported instruction set e.g., "LLVM Instructions."

b) This object only accepts a single child among the following objects:

   1) **FunctionalUnitSet**: Refer to FunctionalUnitSet (6.6.7).

   2) **FunctionalUnitSetFile**: Refer to FunctionalUnitSetFile (6.6.8). This is the recommended way of integrating the *FunctionalUnitSet* in your SHIM system XML.

Example

```
<CommonInstructionSet name="LLVM Instructions">
<FunctionalUnitSet>
<FunctionalUnit name="BR">
    <Instruction operation="ret" outputBitWidth="32" inputBitWidth="32">
        <Performance>
            <Pitch best="10.0" typical="10.0" worst="10.0"/>
            <Latency best="10.0" typical="10.0" worst="10.0"/>
        </Performance>
    </Instruction>
    <Instruction operation="br" outputBitWidth="32" inputBitWidth="32">
        <Performance>
            <Pitch best="10.0" typical="10.0" worst="10.0"/>
            <Latency best="10.0" typical="10.0" worst="10.0"/>
        </Performance>
    </Instruction>
</FunctionalUnit>
</FunctionalUnitSet>
</CommonInstructionSet>
```

[6] http://llvm.org/docs/LangRef.html#instruction-reference

6.6.7 FunctionalUnitSet

Description

This object lists the *FunctionalUnit* that composes the related *MasterComponent.*

Example

See CommonInstructionSet (6.6.6).

6.6.8 FunctionalUnitSetFile

Description

This object provides a reference to an external file defining the *FunctionalUnits* composing the related *MasterComponent*.

— **src** (mandatory; type *string*): The path to the referenced external file, this path is relative to the system XML file.

Example

See *CommonInstructionSet*

```
<CommonInstructionSet name="LLVM Instructions">
    <FunctionalUnitSetFile src="com.arm.cortexa53.llvm.xml" />
<CommonInstructionSet/>
```

6.6.9 FunctionalUnit

Description

This object allows for the defining of the *FucntionalUnits* of the related *MasterComponent*. Each *FunctionalUnit* object lists the *Instructions* and *CustomInstructions* that they support.

— **name** (mandatory; type *string*): The name of the functional unit, it is recommended to use identical names as in the *MasterComponent* technical reference manual.

Example

See CommonInstructionSet (6.6.6).

6.6.10 Instruction

Description

This object describes an instruction supported by the *MasterComponent* in the related *FunctionalUnit*. It has following objects and/or attributes:

— **name** (mandatory; type *string*): The name of this object.

- **operation** (mandatory; type *string*): The name of the operation or instruction in the reference instruction set that is supported by this Instruction (e.g., “add” or “getelementptr” in LLVM instruction set).
- **nInputs** (mandatory; type *nonNegativeInteger*): The number of input of this instruction.
- **nOutputs** (mandatory; type *nonNegativeInteger*): The number of outputs of this instruction.
- **inputBitWidth** (mandatory; type *nonNegativeInteger*): Specifies the bit size of the instruction inputs
- **outputBitWidth** (mandatory; type *nonNegativeInteger*): Specifies the bit size of the instruction output result
- **Performance** (mandatory): Refer to Performance (6.6.15).
- **supportedSignedness** (optional; type *nonNegativeInteger*): Specifies if the instruction supports signed execution, unsigned execution, or both.
- **operandAddressing** (optional; type *OperandAdressingType*): Specifies if the instruction can provide its operands using a register reference, an immediate encoded in the instruction or both.
- **immediateBitWidth** (optional; type *nonNegativeInteger*): If the instruction supports immediate operands (see *operandAdressing*), this attribute specifies the maximum supported immediate bit size.
- **encodingLength** (optional; type *nonNegativeInteger*): Specifies the bit size of the corresponding instruction.
- **SIMDWidth** (optional; type *nonNegativeInteger*): If the instruction supports SIMD execution, this attribute defines its amount of data-parallelism.
- **isEmulated** (optional; type *boolean*): Allows to indicate that the MasterComponent supports the corresponding Instruction operation using software emulation, for instance by using a call to a library (e.g., some CPU do not have floating point ALUs and support floating operations using software libraries).
- **inputPreserved** (optional; type *boolean*): Set to false if the instruction overwrites one of its input register, set to true if the instruction does not change the content of its input register.

Example

See CommonInstructionSet (6.6.6).

6.6.11 CustomInstruction

Description

This object allows a vendor to define an instruction that is supported by the *MasterComponent*, but can only be described by a combination of instructions from the reference instruction set and/or a specific configuration of inputs (e.g., an implied constant value).

A *CustomInstruction* must define its inputs using *InstructionInput* objects, the operations it performs using one or several *InstructionOperation* objects, and its outputs using *InstructionOutput* objects.

- **name** (mandatory; type *string*): The name of this object.
- **nInputs** (mandatory; type *nonNegativeInteger*): The number of inputs of this instruction.
- **nOutputs** (mandatory; type *nonNegativeInteger*): The number of outputs of this instruction.

— **Performance** (mandatory): Refer to Performance (6.6.15).

— **InstructionInput** (optional): Refer to InstructionInput (6.6.12).

— **InstructionOperation** (optional): Refer to InstructionOperation(6.6.13).

— **InstructionOutput**(optional): Refer to InstructionOutput (6.6.14).

Example

```
<CustomInstruction name="FMADD" nInputs="3" nOutputs="1">
    <Performance>
        <Pitch   best="1.66" typical="1.66" worst="1.0" />
        <Latency best="10.0" typical="10.0" worst="10.0" />
    </Performance>
    <!-- Inputs -->
    <InstructionInput id="FMADD_IN_A" bitWidth="32" type="FLOAT"/>
    <InstructionInput id="FMADD_IN_B" bitWidth="32" type="FLOAT"/>
    <InstructionInput id="FMADD_IN_C" bitWidth="32" type="FLOAT"/>
    <!-- Operations -->
    <InstructionOperation id="FMADD_MULT" operation="mult">
        <InstructionOperand>FMADD_IN_B</InstructionOperand>
        <InstructionOperand>FMADD_IN_C</InstructionOperand>
    </InstructionOperation>
    <InstructionOperation id="FMADD_MULT_ADD" operation="add">
        <InstructionOperand>FMADD_IN_A</InstructionOperand>
        <InstructionOperand>FMADD_MULT</InstructionOperand>
    </InstructionOperation>
    <!-- Outputs -->
    <InstructionOutput ref="FMADD_MULT_ADD" bitWidth="32" type="FLOAT" />
</CustomInstruction>
```

6.6.12 InstructionInput

Description

This object allows a vendor to define a *CustomInstruction* input.

— **id** (mandatory; type *string*): The id of this object.

— **bitWidth** (mandatory; type *nonNegativeInteger*): Specifies the bit size of the instruction input

— **type** (mandatory; type *InstructionInputType*): Specifies the type of the input, it can either be INTEGER if the input is an integer, FLOAT if the input is a floating-point value, IMMEDIATE if the value is provided by the instruction or POINTER is the input is a memory pointer.

— **value** (optional, type *string*): If the instruction input type is IMMEDIATE, this attribute allows to provide the immediate value.

Example

See CustomInstruction (6.6.11).

6.6.13 InstructionOperation

Description

This object allows for the defining of a *CustomInstruction* operation. This object can have zero, one or several *InstructionOperands* to define the operands on which it performs the operation. The result of this object can be used as the operand of another *InstructionOperation* object or as a *CustomInstruction* output.

— **id** (mandatory; type *string*): The id of this object.

— **operation** (mandatory; type *string*): Specifies the operation from the reference instruction set provided by this object.

— **InstructionOperand** (optional, type *IDREF*): Specify one or several id references of *InstructionInputs* that are used as this object operands. Alternatively, it can also be the id of another *InstructionOperation* object whose result is then used as this object operand.

Example

See CustomInstruction (6.6.11).

6.6.14 InstructionOutput

Description

This object allows for the defining of a *CustomInstruction* output.

— **bitWidth** (mandatory; type *nonNegativeInteger*): Specifies the bit size of the instruction output.

— **type** (mandatory; type *InstructionOutputType*): Specifies the type of the output, it can either be INTEGER if the output is an integer, FLOAT if the output is a floating-point value or POINTER is the output is a memory pointer.

— **ref** (mandatory; type *IDREF*): The id reference of the value provided by this instruction output. It can either be an *InstructionInput* object id or an *InstructionOperation* object id of the related *CustomInstruction*.

Example

See CustomInstruction (6.6.11).

6.6.15 Performance

Description

This object describes performance. It has the following objects and/or attributes:

— **Latency** (mandatory): Refer to Latency (6.6.16).

— **Pitch** (mandatory): Refer to Pitch (6.6.17).

— **accessTypeRef** (optional; type *IDREF*) a reference to AccessType **id**. This is intended to be used when describing the performance of memory access.

Example

See CommonInstructionSet (6.6.6).

6.6.16 Latency

Description

It has the following objects and/or attributes. Refer to Latency (6.6.16) and Pitch (6.6.17).

— **best** (optional; type *float*): The number of processor cycles for the best-case latency.
— **typical** (mandatory; type *float*): The number of processor cycles for the typical latency.
— **worst** (optional; type *float*): The number of processor cycles for the worst-case latency.

Example

See CommonInstructionSet (6.6.6).

6.6.17 Pitch

Description

It has following objects and/or attributes:

— **best** (optional; type *float*): The number of processor cycles for the best-case pitch.
— **typical** (mandatory; type *float*): The number of processor cycles for the typical pitch.
— **worst** (optional; type *float*): The number of processor cycles for the worst-case pitch.

Example

See CommonInstructionSet (6.6.6).

6.7 FrequencyVoltageSet

Schema

Figure 13 is an example of a schema of FrequencyVoltageSet.

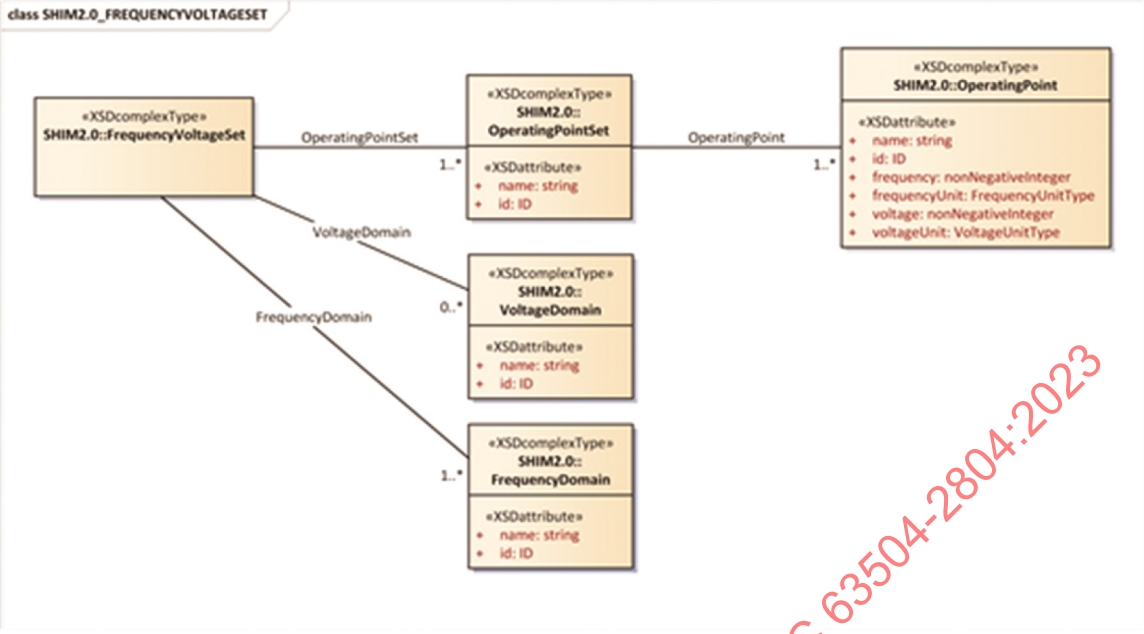


Figure 13—Schema of FrequencyVoltageSet

Description

FrequencyVoltageSet allows to describe the system FrequencyDomains, VoltageDomains, and OperatingPoints.

6.7.1 FrequencyDomain

Description

A *FrequencyDomain* allows for the defining of a group of components that share the same clock frequency.

It has following objects and/or attributes:

— **name** (mandatory; type *string*): The name of this object.

— **id** (mandatory; type *string*): The id of this object.

Example

```
<FrequencyDomain name="FD_CPU_CLUSTER" id="SHIMEDITOR29805129821320141005130145552" />
```

6.7.2 VoltageDomain

Description

A *VoltageDomain* allows a vendor to define a group of components that share the same input voltage.

It has following objects and/or attributes:

— **name** (mandatory; type *string*): The name of this object.

— **id** (mandatory; type *string*): The id of this object.

Example

```
<VoltageDomain name="VD_CPU_CLUSTER" id="SHIMEDITOR2980512982132014100513014555 3" />
```

6.7.3 OperatingPointSet

Description

An *OperatingPointSet* allows a vendor to define a list of valid operating points shared by a set of system components. A SHIM system description must contain at least one *OperatingPointSet* in its *FrequencyVoltageSet*. It has following objects and/or attributes:

- **name** (mandatory; type *string*): The name of this object.
- **id** (mandatory; type *string*): The id of this object.

Example

```
<OperatingPointSet name="OPS_CPU_CLUSTER" id="SHIMEDITOR2980512982132014100513014555 4">
    <OperatingPoint name="OP_CPU_CLUSTER_OFF"
id="SHIMEDITOR2980512982132014100513014555 5"
        frequency="0"    frequencyUnit="MHz" voltage="0"    voltageUnit="mV" />
    <OperatingPoint name="OP_CPU_CLUSTER_LOWSPEED"
id="SHIMEDITOR2980512982132014100513014555 6"
        frequency="1666" frequencyUnit="MHz" voltage="1000" voltageUnit="mV" />
    <OperatingPoint name="OP_CPU_CLUSTER_MAXSPEED"
id="SHIMEDITOR2980512982132014100513014555 7"
        frequency="2100" frequencyUnit="MHz" voltage="1250" voltageUnit="mV" />
</OperatingPointSet>
```

6.7.4 OperatingPoint

Description

An *OperatingPoint* defines a component operating point.

It has following objects and/or attributes:

- **name** (mandatory; type *string*): The name of this object.
- **id** (mandatory; type *string*): The id of this object.
- **frequency** (mandatory; type *nonNegativeInteger*): The operation point clock frequency value.
- **frequencyUnit** (optional; type *FrequencyUnitType*): Define the unit of the related clock frequency value (see *frequency* attribute).
- **voltage** (optional; type *nonNegativeInteger*): The operation point input voltage value.
- **voltageUnit** (optional; type *VoltageUnitType*): Define the unit of the related input voltage value (see *voltage* attribute).

Example

See OperatingPointSet (6.7.3).

6.8 AddressSpaceSet

Schema

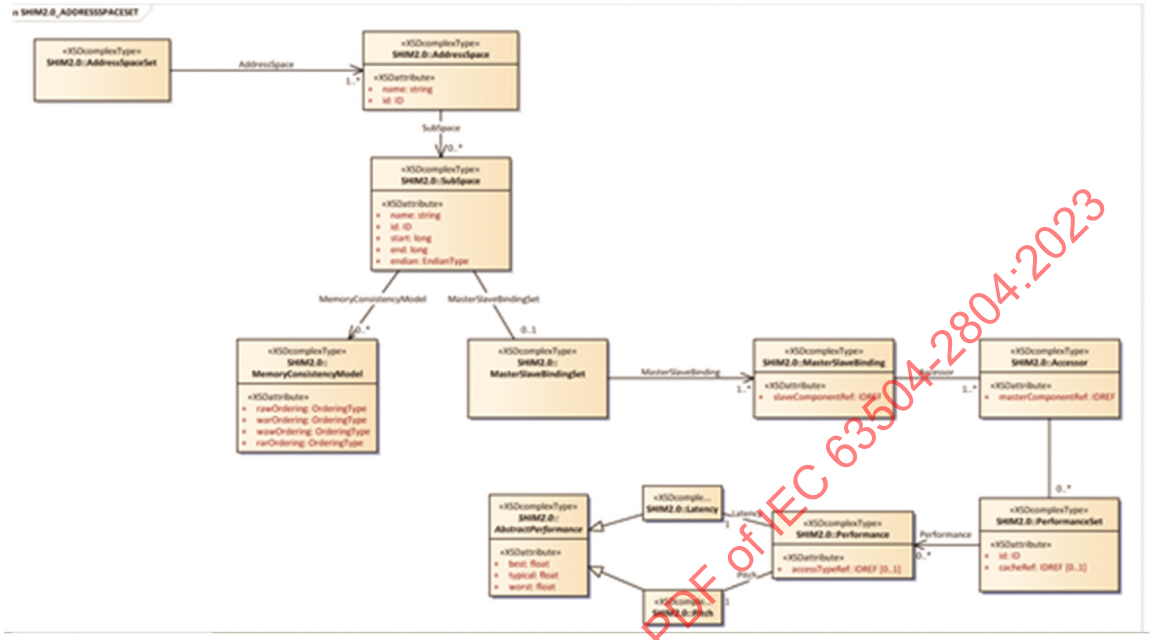


Figure 14—Schema of AddressSpaceSet

Description

AddressSpaceSet describes how the memory address spaces are organized and which *MasterComponent* is bound to which *SlaveComponent.*

6.8.1 AddressSpace

Description

It has following objects and/or attributes:

- **SubSpace** (optional): Refer to SubSpace (6.8.2).
- **name** (mandatory; type *string*): The name of this object.
- **id** (mandatory; type *ID*): The ID of this object.

Example

```
<AddressSpaceSet>
    <AddressSpace name="AS_0_0" id="SHIMEDITOR223752652069201410051301433 54">
        <SubSpace name="SS_0_0_0" id="SHIMEDITOR914093813232014100513014335 5" start="0"
    end="128"
                endian="LITTLE">
            <MemoryConsistencyModel rawOrdering="ORDERED" warOrdering="ORDERED"
                                    wawOrdering="ORDERED" rarOrdering="ORDERED"/>
            <MasterSlaveBindingSet>
                <MasterSlaveBinding
slaveComponentRef="SHIMEDITOR818177486502014100513014297 5">
```

```
                    <Accessor
masterComponentRef="SHIMEDITOR25331849408820141005130142974">
                       <PerformanceSet id="SHIMEDITOR272374223931201410051301455 45"
                                       cacheRef="SHIMEDITOR272374223931201410051301789 45">
                          <Performance
accessTypeRef="SHIMEDITOR272374223931201410051301455 51">
                             <Pitch best="10.0" typical="10.0" worst="10.0"/>
                             <Latency best="10.0" typical="10.0" worst="10.0"/>
                          </Performance>
                          <Performance
accessTypeRef="SHIMEDITOR298051298213201410051301455 52">
                             <Pitch best="10.0" typical="10.0" worst="10.0"/>
                             <Latency best="10.0" typical="10.0" worst="10.0"/>
                          </Performance>
                       </PerformanceSet>
                    </Accessor>
                    ...
                 </MasterSlaveBinding>
                 ...
              </MasterSlaveBindingSet>
           </SubSpace>
           ...

</AddressSpace>
    ...
<AddressSpaceSe
t>
```

6.8.2 SubSpace

Description

This object describes a segment in *AddressSpace*. It has the following objects and/or attributes:

- **MasterSlaveBindingSet** (mandatory): Refer to MasterSlaveBindingSet (6.8.4).
- **MemoryConsistencyModel** (optional): Refer to MemoryConsistencyModel (6.8.3).
- **name** (mandatory; type *string*): The name of this object.
- **id** (mandatory; type *ID*): The ID of this object.
- **start** (mandatory; type *long*): The start address.
- **end** (mandatory; type *long*): The end address.
- **endian** (optional; type *EndianType*): The endianness of this object.

Example

See AddressSpace (6.8.1).

6.8.3 MemoryConsistencyModel

Description

It has the following objects and/or attributes:

- **rawOrdering** (optional; type *OrderingType*): Specifies the memory ordering of read-after-write access.

— **warOrdering** (optional; type *OrderingType*): Specifies the memory ordering of write-after-read access.

— **wawOrdering** (optional; type *OrderingType*): Specifies the memory ordering of write-after-write access.

— **rarOrdering** (optional; type *OrderingType*): Specifies the memory ordering of read-after-read access.

Example

See AddressSpace (6.8.1).

6.8.4 MasterSlaveBindingSet

Description

It has the following objects and/or attributes:

— **MasterSlaveBinding** (mandatory): Refer to MasterSlaveBinding (6.8.5).

Example

See AddressSpace (6.8.1).

6.8.5 MasterSlaveBinding

Description

This object binds a *MasterComponent* to a *SlaveComponent*. It has the following objects and/or attributes:

— **Accessor** (mandatory): Refer to Accessor (6.8.6).

— **slaveComponentRef** (mandatory; type *IDREF*): Specifies the **id** of *SlaveComponent.*

Example

See AddressSpace (6.8.1).

6.8.6 Accessor

Description

It has the following objects and/or attributes:

— **PerformanceSet** (optional): Refer to PerformanceSet (6.8.7).

— **masterComponentRef** (mandatory; type *IDREF*): Specifies the **id** of *MasterComponent*.

Example

See AddressSpace (6.8.1).

6.8.7 PerformanceSet

Description

This groups one or more Performance (6.6.15). It has the following objects and/or attributes:

— **id** (mandatory, type *ID*): Defines the id of this object.

— **cacheRef** (optional, type *IDREF*): This attribute allows a vendor to specify a cache that intercepts the related memory access, i.e., the cache where the related memory access produces a cache hit.

— **Performance** (optional): Refer to Performance (6.6.15).

Example

See AddressSpace (6.8.1).

6.9 CommunicationSet

Schema

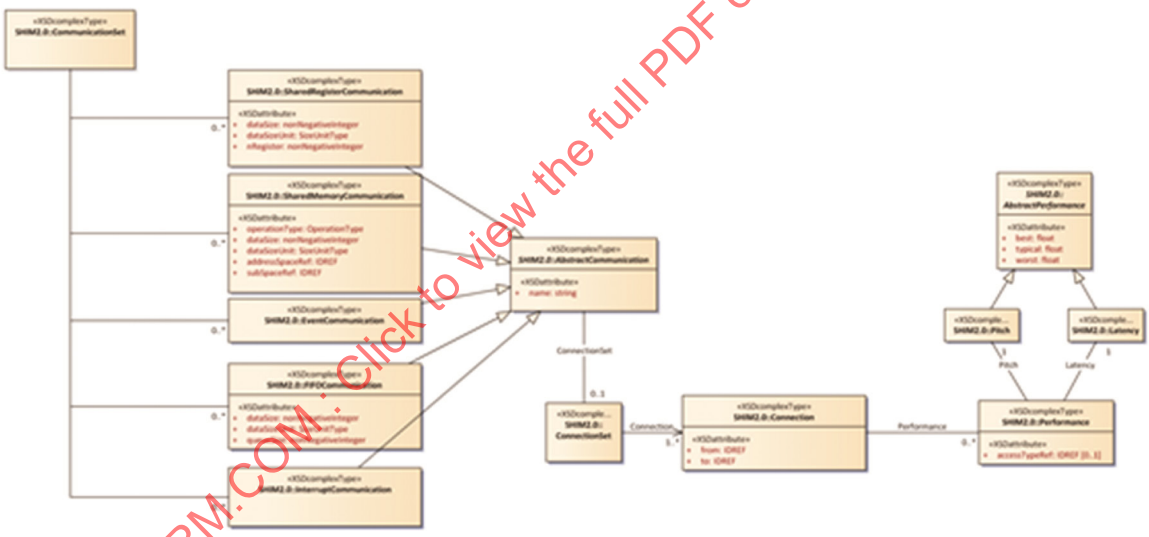


Figure 15—Schema of CommunicationSet

Description

CommunicationSet describes the available *MasterComponent*-to-*MasterComponent* communication. There are six objects that describe different types of communication.

6.9.1 FIFOCommunication

Description

This object describes FIFO-based communication. It has the following objects and/or attributes:

— **ConnectionSet** (mandatory): Refer to ConnectionSet (6.9.6).

— **name** (mandatory; type *string*): The name of this object.

— **dataSize** (mandatory; type *nonNegativeInteger*): The data size of this FIFO.

— **dataSizeUnit** (mandatory; type *SizeUnitType*): The unit of **dataSize** of this FIFO.

— **queueSize** (mandatory; type *nonNegativeInteger*): The queue size (multiples of dataSize in dataSizeUnit, so that the total capacity is a product of dataSize * dataSizeUnit * queueSize) of this FIFO.

Example

```
<FIFOCommunication dataSize="64" dataSizeUnit="KiB" queueSize="32" name="fifo_00">
<ConnectionSet>
        <Connection from="SHIMEDITOR25331849408820141005130142974"
                    to="SHIMEDITOR31113490118120141005130142974">
            <Performance>
                <Pitch best="10.0" typical="10.0" worst="10.0"/>
                <Latency best="10.0" typical="10.0" worst="10.0"/>
            </Performance>
        </Connection>
         ...
</ConnectionSet>
</FIFOCommunication>
```

6.9.2 SharedRegisterCommunication

Description

This object describes a shared-register based communication. It has the following objects and/or attributes:

— **ConnectionSet** (mandatory): Refer to ConnectionSet (6.9.6).

— **name** (mandatory; type *string*): The name of this object.

— **dataSize** (mandatory; type *nonNegativeInteger*): The data size of one shared register.

— **dataSizeUnit** (mandatory; type *SizeUnitType*): The unit of **dataSize** of this shared register.

— **nRegister** (mandatory; type *nonNegativeInteger*): The number of shared registers.

Example

```
<SharedRegisterCommunication dataSize="32" dataSizeUnit="KiB" nRegister="32"
name="sreg_00">     <ConnectionSet>
        <Connection from="SHIMEDITOR25331849408820141005130142974"
                    to="SHIMEDITOR31113490118120141005130142974">
            <Performance>
                <Pitch best="10.0" typical="10.0" worst="10.0"/>
                <Latency best="10.0" typical="10.0" worst="10.0"/>
            </Performance>
        </Connection>
        ...
</ConnectionSet>
</SharedRegisterCommunication>
```

6.9.3 InterruptCommunication

Description

It has the following objects and/or attributes:

— **ConnectionSet** (mandatory): Refer to ConnectionSet (6.9.6).

— **name** (mandatory; type *string*): The name of this object.

Example

```
<InterruptCommunication name="Interrupt_00">
    <ConnectionSet>
        <Connection from="SHIMEDITOR2533184940882014100513014297 4"
                    to="SHIMEDITOR3111349011812014100513014297 4">
            <Performance>
                <Pitch best="10.0" typical="10.0" worst="10.0"/>
                <Latency best="10.0" typical="10.0" worst="10.0"/>
            </Performance>
        </Connection>
    </ConnectionSet>
</InterruptCommunication>
```

6.9.4 SharedMemoryCommunication

Description

It has the following objects and/or attributes:

— **ConnectionSet** (mandatory): Refer to ConnectionSet (6.9.6).

— **name** (mandatory; type *string*): The name of this object.

— **operationType** (optional; type *OperationType*): The type of this shared memory communication.

— **dataSize** (optional; type *nonNegativeInteger*): The data size of this *SharedMemoryCommunication*.

— **dataSizeUnit** (mandatory; type *SizeUnitType*): The unit of **dataSize** of this shared memory.

— **addressSpaceRef** (optional; type *IDREF*): Specifies the **id** of AddressSpace (6.8.1) that the shared memory this object uses. If this attribute is not declared and the subsequent *subSpaceRef* is not declared, then it means the *SharedMemoryCommunication* mechanism is valid for all *AddressSpace* and *SubSpace*.

— **subSpaceRef** (optional; type *IDREF*): Specifies the **id** of SubSpace (6.8.2) that this *SharedMemoryCommunication* supports. When this attribute is declared, the corresponding *addressSpaceRef* must also be declared as above. When this attribute is omitted and *addressSpaceRef* is declared, it means the *SharedMemoryCommunication* mechanism is valid for all *SubSpace* for the declared *AddressSpace*. When *addressSpaceRef* is omitted but *subSpaceRef* is declared, the interpretation is undefined and must not be used.

Example

```
<SharedMemoryCommunication dataSize="128" dataSizeUnit="KiB"
     addressSpaceRef="SHIMEDITOR2237526520692014100513014335 4"
     subSpaceRef="SHIMEDITOR9140938132320141005130143355" name="shmem_00">
    <ConnectionSet>
        <Connection from="SHIMEDITOR2533184940882014100513014297 4"
                    to="SHIMEDITOR3111349011812014100513014297 4">
            <Performance>
                <Pitch best="10.0" typical="10.0" worst="10.0"/>
                <Latency best="10.0" typical="10.0" worst="10.0"/>
              </Performance>
```

```
        </Connection>
        ...
    </ConnectionSet>
</SharedMemoryCommunication>
```

6.9.5 EventCommunication

Description

This object describes an event-based communication. It has the following objects and/or attributes:

— **ConnectionSet** (mandatory;): Refer to ConnectionSet (6.9.6).

— **name** (mandatory; type *string*): The name of this object.

Example

```
<EventCommunication name="Event_00">
    <ConnectionSet>
        <Connection from="SHIMEDITOR2533184940882014100513014297A"
                    to="SHIMEDITOR3111349011812014100513014297A">
            <Performance>
                <Pitch best="10.0" typical="10.0" worst="10.0"/>
                <Latency best="10.0" typical="10.0" worst="10.0"/>
            </Performance>
        </Connection>
        ...
    </ConnectionSet>
</EventCommunication>
```

6.9.6 ConnectionSet

Description

It has the following objects and/or attributes:

— **Connection** (mandatory): Refer to Connection (6.9.7).

Example

See examples in various communication objects.

6.9.7 Connection

Description

It has the following objects and/or attributes:

— **Performance** (optional): Refer to Performance (6.6.15).

— **from** (mandatory; type *IDREF*): The **id** of *MasterComponent* that forms the initiator of the connection.

— **to** (mandatory; type *IDREF*): The **id** of *MasterComponent* that forms the terminal of connection.

Example

See examples in various communication objects.

6.10 ContentionGroupSet

Schema

Figure 16—Schema of ContentionGruopSet

Description

ContentionGroupSet describes one or more *ContentionGroup* present in the system.

Example

```
<ContentionGroupSet>
<ContentionGroup name="axi_bus_0" id="CG_SHAREDBUS_CLUSTER">
    <Throughput value="16" unit="B/cycle" frequencyDomainRef="FD_GENPLAT_CLUSTER" />
    <PerformanceSetRef>SHIMEDITOR25331849408820141005130142974</PerformanceSetRef>
    <PerformanceSetRef>SHIMEDITOR25331849408820141005130142920</PerformanceSetRef>
    <PerformanceSetRef>SHIMEDITOR25331849408820141005130142954</PerformanceSetRef>
    <PerformanceSetRef>SHIMEDITOR25331849408820141005130142978</PerformanceSetRef>
</ContentionGroup>
<ContentionGroup name="axi_bus_1" id="CG_SHAREDBUS_ACCELERATORS">
    <DataRate value="1" unit="GiB/s"/>
    <PerformanceSetRef>SHIMEDITOR25331849408820141005130174132</PerformanceSetRef>
    <PerformanceSetRef>SHIMEDITOR25331849408820141005130120478</PerformanceSetRef>
</ContentionGroup>
</ContentionGroupSet>
```

6.10.1 ContentionGroup

Description

A *ContentionGroup* defines a communication resource that is used by *MasterComponent* accesses to *SlaveComponent*s. This element contains one or several references to the *PerformanceSet* that uses this communication resource. The maximum bandwidth available in this communication resource can be optionally defined by a *Throughput* or *DataRate* object.

It has the following objects and/or attributes:

— **Throughput** (optional): Refer to Throughput (6.10.2).

— **DataRate** (optional): Refer to DataRate (6.10.3).

— **id** (mandatory; type *ID*): The id of this object.

— **name** (mandatory; type *string*): The name of this object, it is recommended to use identical names as in the system technical reference manual.

— **PerformanceSetRef** (mandatory; type *IDREF*): This object allows a vendor to specify the id reference of one or several PerformanceSet objects that make use of this contention group.

Example

See ContentionGroupSet (6.10).

6.10.2 Throughput

Description

A *Throughput* object defines a throughput value and its related *FrequencyDomain*.

It has the following objects and/or attributes:

— **value** (mandatory; type *nonNegativeInteger*): The throughput value

— **unit** (optional; type *ThroughputUnitType*): This attribute allows a vendor to specify the throughput units.

— **frequencyDomainRef** (required, type *IDREF*): This attribute allows a vendor to specify the id of the frequency domain related to this object.

Example

See ContentionGroupSet (6.10).

6.10.3 DataRate

Description

A *DataRate* object defines a data rate value.

It has the following objects and/or attributes:

— **value** (mandatory; type *nonNegativeInteger*): The data rate value

— **unit** (optional; type *DataRateUnitType*): This attribute allows specifying the data rate units.

Example

See ContentionGroupSet (6.10).

6.11 PowerConfiguration

Schema

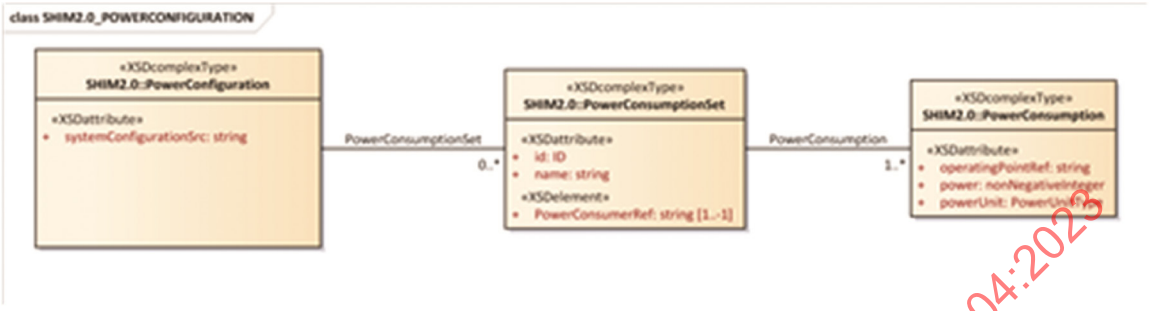


Figure 17—Schema of PowerConfiguration

Description

PowerConfiguration allows for the defining of the power consumption of one or several SHIM system components. It contains one or more *PowerConsumptionSet* elements.

This element is a root object. It is not part of the SHIM 2.0 system file and should be hosted in a separate XML file.

It has the following objects and/or attributes:

— **systemConfigurationSrc** (mandatory; type *string*): The path pointing to the related SHIM 2.0 system XML file. This path is relative to the current XML file.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<shim:Shim
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:shim="http://www.multicore-association.org/2017/SHIM2.0/"
    xsi:schemaLocation="http://www.multicore-association.org/2017/SHIM2.0/ shim20.xsd"
    name="MySystemPower" shimVersion="2.0">
<PowerConfiguration systemConfigurationSrc="com.vendor.mysystem.system.xml">
    <PowerConsumptionSet>
        <PowerConsumerRef>CS_CPU_CLUSTER</PowerConsumerRef>
        <PowerConsumption operatingPointRef="OP_CPU_CLUSTER_OFF"
            power="10" powerUnit="mW"/>
        <PowerConsumption operatingPointRef="OP_CPU_CLUSTER_LOWSPEED"
            power="176" powerUnit="mW"/>
        <PowerConsumption operatingPointRef="OP_CPU_CLUSTER_MAWSPEED"
            power="510" powerUnit="mW"/>
    </PowerConsumptionSet>
</PowerConfiguration>
</shim:Shim>
```

6.11.1 PowerConsumptionSet

Description

A *PowerConsumptionSet* defines the power consumption points of one or several system components. This element contains one or more *PowerConsumption* elements that define the power consumption of their

corresponding *OperatingPoint*. It also contains one or more *PowerConsumerRef*, which points to the system components sharing those *PowerConsumption* definitions.

It has the following objects and/or attributes:

— **id** (mandatory; type *ID*): The id of this object.

— **name** (mandatory; type *string*): The name of this object.

— **powerConsumerRef** (mandatory; type *IDREF*): This object allows a vendor to specify the id reference of one or several system components that share the same *PowerConsumptionSet*.

Example

See PowerConfiguration (6.11.2).

6.11.2 PowerConsumption

Description

A *PowerConsumption* defines the power consumption associated with an *OperatingPoint*. It has the following objects and/or attributes:

— **id** (mandatory; type *ID*): The id of this object.

— **name** (mandatory; type *string*): The name of this object.

— **operatingPointRef** (mandatory; type *IDREF*): This object allows a vendor to specify the id reference of the *OperatingPoint* related to this power consumption definition.

— **power** (optional; type *nonNegativeInteger*): This attribute allows a vendor to specify the power consumption value.

— **powerUnit** (optional; type *PowerUnitType*): This attribute allows a vendor to specify the power units of the power attribute.

Example

See PowerConfiguration (6.11).

6.12 VendorExtension

Schema

Figure 18—Schema of VendorExtension

Description

VendorExtension allows a vendor to define a set of extension to the standard SHIM 2.0 system file and SHIM 2.0 power files.

This element is a root object. It is not part of the SHIM 2.0 system file and should be hosted in a separate XML file. This element can contain one or several references to other SHIM2.0 system and power files.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<shim:Shim
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:shim="http://www.multicore-association.org/2017/SHIM2.0/"
    xsi:schemaLocation="http://www.multicore-association.org/2017/SHIM2.0/ shim20.xsd"
    name="MyVendorExt" shimVersion="2.0">
    <VendorExtension>
        <SystemConfigurationFile src="com.vendor.mysystem.system.xml" />
        <PowerConfigurationFile src="com.vendor.mysystem.power.xml" />
    </VendorExtension>
</shim:Shim>
```

6.12.1 SystemConfigurationFile

Description

SystemConfigurationFile allows a vendor to define a reference to a SHIM2.0 system XML file.

It has the following objects and/or attributes:

- **id** (mandatory; type *ID*): The id of this object.
- **name** (mandatory; type *string*): The name of this object.
- **src** (mandatory; type *string*): The path to a SHIM2.0 system XML file. The path is relative to the directory hosting the vendor extension XML file.

Example

See VendorExtension (6.12).

6.12.2 PowerConfigurationFile

Description

PowerConfigurationFile allows a vendor to define a reference to a SHIM2.0 power XML file.

It has the following objects and/or attributes:

- **id** (mandatory; type *ID*): The id of this object.
- **name** (mandatory; type *string*): The name of this object.
- **src** (mandatory; type *string*): The path to a SHIM2.0 power XML file. The path is relative to the directory hosting the vendor extension XML file.

Example

See VendorExtension (6.12).

7. Use cases

These use cases are provided as examples to see how to use the information that SHIM XML contains. The categories of tools mentioned are to exemplify and to help the user to understand the concepts. This may also apply to other types of tools.

7.1 Performance estimation: Auto-parallelizing compiler

Table 7 shows performance estimation.