BS IEC 61691-8:2021



BSI Standards Publication

Behavioural languages

Part 8: Standard SystemC® Analog/Mixed-Signal Extensions Language Reference Manual

bsi.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

National foreword

This British Standard is the UK implementation of IEC 61691-8:2021.

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

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

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

© The British Standards Institution 2021
Published by BSI Standards Limited 2021

ISBN 978 0 539 17206 5

ICS 25.040.01; 35.060

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

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 August 2021.

Amendments/corrigenda issued since publication

Date	Text affected

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

IEC 61691-8

Edition 1.0 2021-07

INTERNATIONAL STANDARD IEEE Std 1666.1™

Behavioural languages – Part 8: Standard SystemC® Analog/Mixed-Signal Extensions Language Reference Manual

INTERNATIONAL ELECTROTECHNICAL COMMISSION

ICS 25.040.01; 35.060

ISBN 978-2-8322-9951-7

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

® Registered trademark of the International Electrotechnical Commission

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

Contents

1. Overview 1

1.1 Scope 1
1.2 Purpose 1
1.3 Subsets 2
1.4 Relationship with C++ 2
1.5 Relationship with SystemC 2
1.6 Guidance for readers 2

2. Normative references 4

3. Terminology and conventions used in this standard 5

3.1 Terminology 5
3.1.1 Shall, should, may, can 5
3.1.2 Implementation, application 5
3.1.3 Call, called from, derived from 5
3.1.4 Specific technical terms 5
3.2 Syntactical conventions 6
3.2.1 Implementation-defined 6
3.2.2 Disabled 6
3.2.3 Ellipsis (...) 6
3.2.4 Class names 6
3.2.5 Prefixes 7
3.3 Typographical conventions 7
3.4 Semantic conventions 7
3.4.1 Class definitions and the inheritance hierarchy 7
3.4.2 Function definitions and side-effects 7
3.4.3 Functions whose return type is a reference or a pointer 8
3.4.4 Namespaces and internal naming 8
3.4.5 Non-compliant applications and errors 9
3.5 Notes and examples 9

4. Core language definitions 10

4.1 Class header files 10
4.1.1 #include “systemc-ams” 10
4.1.2 #include “systemc-ams.h” 10
4.2 Base class definitions 11
4.2.1 sca_core::sca_module 11
4.2.2 sca_core::sca_interface 13
4.2.3 sca_core::sca_prim_channel 14

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

AVA Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

4.2.4 sca_core::sca_port....15
4.2.5 sca_core::sca_time....16
4.2.6 sca_core::sca_max_time....16
4.2.7 sca_core::sca_parameter_base....16
4.2.8 sca_core::sca_parameter....18
4.2.9 sca_core::*sca_assign_from_proxy*†....20
4.2.10 sca_core::*sca_assign_to_proxy*† 21

5. Timed data flow model of computation....22

5.1 Class definitions....22
5.1.1 sca_tdf::sca_module....22
5.1.2 sca_tdf::sca_signal_if....29
5.1.3 sca_tdf::sca_signal....30
5.1.4 sca_tdf::sca_default_interpolator....31
5.1.5 sca_tdf::sca_in....32
5.1.6 sca_tdf::sca_out....37
5.1.7 sca_tdf::sca_out<T>....39
5.1.8 sca_tdf::sca_out<T, sca_tdf::SCA_CT_CUT, INTERP>....44
5.1.9 sca_tdf::sca_out<T, sca_tdf::SCA_DT_CUT>....50
5.1.10 sca_tdf::sca_de::sca_in, sca_tdf::sc_in....55
5.1.11 sca_tdf::sca_de::sca_in<bool>, sca_tdf::sc_in<bool>....62
5.1.12 sca_tdf::sca_de::sca_in<sc_dt::sc_logic>, sca_tdf::sc_in<sc_dt::sc_logic>....69
5.1.13 sca_tdf::sca_de::sca_out, sca_tdf::sc_out....75
5.1.14 sca_tdf::sca_trace_variable....82
5.2 Hierarchical composition and port binding....83
5.3 Elaboration and simulation....83
5.3.1 Elaboration....84
5.3.2 Simulation....85
5.4 Embedded linear dynamic equations....87
5.4.1 sca_tdf::*sca_ct_proxy*†....88
5.4.2 sca_tdf::*sca_ct_vector_proxy*†....89
5.4.3 sca_tdf::sca_ltf_nd....91
5.4.4 sca_tdf::sca_ltf_zp....96
5.4.5 sca_tdf::sca_ss....102

6. Linear signal flow model of computation....110

6.1 Class definitions....110
6.1.1 sca_lsf::sca_module....110
6.1.2 sca_lsf::sca_signal_if....110
6.1.3 sca_lsf::sca_signal....111
6.1.4 sca_lsf::sca_in....111
6.1.5 sca_lsf::sca_out....112

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

6.1.6 sca_lsf::sca_add ... 113
6.1.7 sca_lsf::sca_sub ... 114
6.1.8 sca_lsf::sca_gain ... 115
6.1.9 sca_lsf::sca_dot ... 115
6.1.10 sca_lsf::sca_integ ... 116
6.1.11 sca_lsf::sca_delay ... 117
6.1.12 sca_lsf::sca_source ... 118
6.1.13 sca_lsf::sca_ltf_nd ... 119
6.1.14 sca_lsf::sca_ltf_zp ... 120
6.1.15 sca_lsf::sca_ss ... 121
6.1.16 sca_lsf::sca_tdf::sca_gain, sca_lsf::sca_tdf_gain ... 122
6.1.17 sca_lsf::sca_tdf::sca_source, sca_lsf::sca_tdf_source ... 123
6.1.18 sca_lsf::sca_tdf::sca_sink, sca_lsf::sca_tdf_sink ... 124
6.1.19 sca_lsf::sca_tdf::sca_mux, sca_lsf::sca_tdf_mux ... 125
6.1.20 sca_lsf::sca_tdf::sca_demux, sca_lsf::sca_tdf_demux ... 126
6.1.21 sca_lsf::sca_de::sca_gain, sca_lsf::sca_de_gain ... 127
6.1.22 sca_lsf::sca_de::sca_source, sca_lsf::sca_de_source ... 127
6.1.23 sca_lsf::sca_de::sca_sink, sca_lsf::sca_de_sink ... 128
6.1.24 sca_lsf::sca_de::sca_mux, sca_lsf::sca_de_mux ... 129
6.1.25 sca_lsf::sca_de::sca_demux, sca_lsf::sca_de_demux ... 130
6.2 Hierarchical composition and port binding ... 131
6.3 Elaboration and simulation ... 131
6.3.1 Elaboration ... 131
6.3.2 Simulation ... 132

7. Electrical linear networks model of computation ... 134

7.1 Class definitions ... 134
7.1.1 sca_eln::sca_module ... 134
7.1.2 sca_eln::sca_node_if ... 135
7.1.3 sca_eln::sca_terminal ... 135
7.1.4 sca_eln::sca_node ... 136
7.1.5 sca_eln::sca_node_ref ... 137
7.1.6 sca_eln::sca_r ... 138
7.1.7 sca_eln::sca_c ... 138
7.1.8 sca_eln::sca_l ... 139
7.1.9 sca_eln::sca_vcvs ... 140
7.1.10 sca_eln::sca_vccs ... 141
7.1.11 sca_eln::sca_ccvs ... 142
7.1.12 sca_eln::sca_cccs ... 142
7.1.13 sca_eln::sca_nullor ... 143
7.1.14 sca_eln::sca_gyrator ... 144
7.1.15 sca_eln::sca_ideal_transformer ... 145

Documents
Document Sharing Hub
Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Full document details: BS IEC 61691-8:2021

7.1.16 sca_eln::sca_transmission_line....146
7.1.17 sca_eln::sca_vsource....147
7.1.18 sca_eln::sca_isource....148
7.1.19 sca_eln::sca_tdf::sca_r, sca_eln::sca_tdf_r....149
7.1.20 sca_eln::sca_tdf::sca_c, sca_eln::sca_tdf_c....150
7.1.21 sca_eln::sca_tdf::sca_l, sca_eln::sca_tdf_l....151
7.1.22 sca_eln::sca_tdf::sca_rswitch, sca_eln::sca_tdf_rswitch....152
7.1.23 sca_eln::sca_tdf::sca_vsource, sca_eln::sca_tdf_vsource....153
7.1.24 sca_eln::sca_tdf::sca_isource, sca_eln::sca_tdf_isource....154
7.1.25 sca_eln::sca_tdf::sca_vsink, sca_eln::sca_tdf_vsink....155
7.1.26 sca_eln::sca_tdf::sca_isink, sca_eln::sca_tdf_isink....156
7.1.27 sca_eln::sca_de::sca_r, sca_eln::sca_de_r....157
7.1.28 sca_eln::sca_de::sca_c, sca_eln::sca_de_c....158
7.1.29 sca_eln::sca_de::sca_l, sca_eln::sca_de_l....159
7.1.30 sca_eln::sca_de::sca_rswitch, sca_eln::sca_de_rswitch....160
7.1.31 sca_eln::sca_de::sca_vsource, sca_eln::sca_de_vsource....161
7.1.32 sca_eln::sca_de::sca_isource, sca_eln::sca_de_isource....162
7.1.33 sca_eln::sca_de::sca_vsink, sca_eln::sca_de_vsink....162
7.1.34 sca_eln::sca_de::sca_isink, sca_eln::sca_de_isink....163
7.2 Hierarchical composition and port binding....164
7.3 Elaboration and simulation....164
7.3.1 Elaboration....165
7.3.2 Simulation....165

8. Predefined analyses....167

8.1 Time-domain analysis....167
8.1.1 Elaboration and simulation....167
8.1.2 Running elaboration and simulation....167
8.2 Small-signal frequency-domain analyses....167
8.2.1 Elaboration and simulation....168
8.2.2 Running elaboration and simulation....168
8.2.3 Small-signal frequency-domain analysis of TDF descriptions....169
8.2.4 Small-signal frequency-domain analysis of LSF descriptions....173
8.2.5 Small-signal frequency-domain analysis of ELN descriptions....173

9. Utility definitions....174

9.1 Trace files....174
9.1.1 Class definitions....174
9.1.2 Function declarations....177
9.2 Data types and constants....180
9.2.1 Class definition and function declarations....180
9.2.2 Definition of constants....188

Documents
Document Sharing Hub
Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Full document details: BS IEC 61691-8:2021

9.3 Reporting information 189
9.3.1 Class definition and function declarations 189
9.3.2 Mask definitions 190
9.4 Version and copyright 191
9.4.1 Macro definitions 191
9.4.2 Constants 193
9.4.3 Function declarations 193

Annex A (informative) Introduction to the SystemC Analog/Mixed-Signal extensions 195

Annex B (informative) Glossary 208

Annex C (informative) Participants 210

Index 211

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

Behavioural languages – Part 8: Standard SystemC® Analog/Mixed-Signal Extensions Language Reference Manual

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* http://standards.ieee.org/ipr/disclaimers.html *for more information).*

   IEC collaborates closely with IEEE in accordance with conditions determined by agreement between the two organizations. 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 61691-8/IEEE Std 1666.1 was processed through IEC technical committee 91: Electronics assembly technology, under the IEC/IEEE Dual Logo Agreement. It is an International Standard.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

AVA Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

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

IEEE Std	FDIS	Report on voting
1666.1 (2016)	91/1712/FDIS	91/1724/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,
- replaced by a revised edition, or
- amended.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

IEEE Standard for Standard SystemC® Analog/Mixed-Signal Extensions Language Reference Manual

Sponsor

Design Automation Standards Committee
of the
IEEE Computer Society

Approved 29 January 2016

IEEE-SA Standards Board

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

Abstract: The SystemC® Analog/Mixed-Signal (AMS) extensions are defined in this standard. SystemC AMS is an ANSI standard C++ class library for electronic system-level design and modeling for use by system architects and engineers who need to address complex heterogeneous systems that are a hybrid between analog, digital and software components. This standard provides a precise and complete definition of the SystemC AMS class library so that a SystemC AMS implementation can be developed with reference to this standard alone. The primary audiences for this standard are the implementors of the SystemC AMS class library, the implementors of tools supporting the class library, and the users of the class library.

Keywords: analog mixed signal, behavioral modeling, C++, computer languages, data flow simulation, digital systems, discrete event simulation, electronic design automation, electronic system level, electronic systems, electrical networks, hardware description language, hardware design, hardware verification, IEEE 1666™, IEEE 1666.1™, mixed-signal modeling, SystemC, SystemC AMS, signal flow modeling, system modeling, system-on-chip

Acknowledgment: Grateful acknowledgment is made to the Accellera Systems Initiative for the permission to use the following source material: Standard SystemC® AMS extensions 2.0 Language Reference Manual.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

Important Notices and Disclaimers Concerning IEEE Standards Documents

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

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

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

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

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

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

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

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

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.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

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, noncommercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Updating of IEEE Standards documents

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

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

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

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

Introduction

This introduction is not part of IEEE Std 1666.1™-2016, IEEE Standard for Standard SystemC® Analog/Mixed-Signal Extensions Language Reference Manual.

This document defines the SystemC Analog/Mixed-Signal (AMS) extensions, which is a C++ class library.

As the electronics industry builds more complex heterogeneous systems involving large numbers of components including analog, digital and software, there is an increasing need for a modeling language that can manage the complexity, heterogeneity, and size of these systems. SystemC AMS provides a mechanism for managing this complexity with its facility for modeling hardware and software together at multiple levels of abstraction. This capability is not available in traditional hardware description languages.

Stakeholders in SystemC AMS include Electronic Design Automation (EDA) companies who implement SystemC AMS class libraries and tools, integrated circuit (IC) suppliers who extend those class libraries and use SystemC AMS to model their intellectual property, and end users who use SystemC AMS to model their systems.

This standard is not intended to serve as a user's guide or to provide an introduction to SystemC AMS. Readers requiring a SystemC AMS tutorial or information on the intended use of SystemC AMS should consult the Accellera Systems Initiative Web site (http://www.accellera.org) to locate the supplemental material and training classes available.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

IEEE Standard for Standard SystemC® Analog/Mixed-Signal Extensions Language Reference Manual

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

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

1. Overview

1.1 Scope

This standard defines the Analog/Mixed-Signal extensions for SystemC®[1], as an ANSI standard C++ class library based on SystemC for system and hardware design including analog/mixed-signal elements.

1.2 Purpose

The general purpose of the SystemC AMS extensions is to provide a C++ standard for designers and architects, who need to address complex heterogeneous systems that are a hybrid between hardware and software. This standard is built on the IEEE Std 1666™-2011[2] (SystemC Language Reference Manual) and extends it to create analog/mixed-signal, multi-disciplinary models to simulate continuous-time, discrete-time, and discrete-event behavior simultaneously.

The specific purpose of this standard is to provide a precise and complete definition of the AMS class library, so that a SystemC AMS implementation can be developed with reference to this standard alone. This standard is neither intended to serve as a user's guide nor to provide an introduction to AMS extensions in SystemC, but does contain useful information for end users.

[1] SystemC® is a registered trademark of the Accellera Systems Initiative.
[2] Information on references can be found in Clause 2.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

1.3 Subsets

It is anticipated that tool vendors will create implementations that support only a subset of this standard or that impose further constraints on the use of this standard. Such implementations are not fully compliant with this standard but may nevertheless claim partial compliance with this standard and may use the name SystemC AMS extensions.

1.4 Relationship with C++

This standard is closely related to the C++ programming language and adheres to the terminology used in ISO/IEC 14882:2003. This standard does not seek to restrict the usage of the C++ programming language; an application using the SystemC AMS extensions may use any of the facilities provided by C++, which in turn may use any of the facilities provided by C. However, where the facilities provided by this standard are used, they shall be used in accordance with the rules and constraints set out in this standard.

This standard defines the public interface to the SystemC AMS class library and the constraints on how those classes may be used. The SystemC AMS class library may be implemented in any manner whatsoever, provided only that the obligations imposed by this standard are honored.

A C++ class library may be extended using the mechanisms provided by the C++ language. Implementors and users are free to extend SystemC AMS extensions in this way, provided that they do not violate this standard.

NOTE—It is possible to create a well-formed C++ program that is legal according to the C++ programming language standard but that violates this standard. An implementation is not obliged to detect every violation of this standard.[3]

1.5 Relationship with SystemC

This standard is built on IEEE Std 1666-2011 and extends it using the mechanisms provided by the C++ language, to provide an additional layer of analog/mixed-signal constructs. Consequently, an implementation and application may use the SystemC core language and predefined channels defined in the namespace **sc_core** and the SystemC data types defined in the namespace **sc_dt**, unless stated otherwise.

Any SystemC compliant application shall behave the same in the presence of the SystemC AMS extensions.

1.6 Guidance for readers

Readers who are not entirely familiar with the SystemC AMS extensions should start with Annex A, which provides a brief informal summary of the subject intended to aid in the understanding of the normative definitions. Such readers may also find it helpful to scan the examples embedded in the normative definitions and to see Annex B.

Readers should pay close attention to Clause 3. An understanding of the terminology and conventions defined in that clause is necessary for a precise interpretation of this standard.

The semantic definitions given in the subsequent clauses detailing the individual classes are built upon the foundations laid in Clause 4.

Clause 5, Clause 6 and Clause 7 define the public interface to the SystemC AMS class library defining the predefined models of computation.

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

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

The following information is listed for each class:

a) A brief class description.

b) A C++ source code listing of the class definition.

c) A statement of any constraints on the use of the class and its members.

d) A statement of the semantics of the class and its members.

e) For certain classes, a description of functions, typedefs, macros, and template parameters associated with the class.

For each predefined model of computation, the execution semantics for elaboration and simulation are defined.

Readers should bear in mind that the primary obligation of a tool vendor is to implement the abstract semantics defined in Clause 5, Clause 6, and Clause 7, using the framework and constraints provided by the class definitions starting in Clause 4.

Annex A is intended to aid the reader in the understanding of the structure and intent of the SystemC AMS class library.

Annex B is giving informal descriptions of the terms used in this standard.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

2. Normative references

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

This standard shall be used in conjunction with the following publications:

ISO/IEC 14882:2003, Programming Languages—C++[4]

IEEE Std 1666™-2011: IEEE Standard for Standard SystemC Language Reference Manual[5,6]

[4]ISO/IEC publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse (http://www.iso.ch/). ISO/IEC publications are also available in the United States from Global Engineering Documents, 15 Inverness Way East, Englewood, Colorado 80112, USA (http://global.ihs.com/). Electronic copies are available in the United States from the American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/).

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

[6]IEEE publications are available from the Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (http://standards.ieee.org/).

Documents
Document Sharing Hub
Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Full document details: BS IEC 61691-8:2021

3. Terminology and conventions used in this standard

3.1 Terminology

3.1.1 Shall, should, may, can

The word *shall* is used to indicate a mandatory requirement.

The word *should* is used to recommend a particular course of action, but does not impose any obligation.

The word *may* is used to mean shall be permitted (in the sense of being legally allowed).

The word *can* is used to mean shall be able to (in the sense of being technically possible).

In some cases, word usage is qualified to indicate on whom the obligation falls, such as *an application may* or *an implementation shall*.

3.1.2 Implementation, application

The word *implementation* is used to mean any specific implementation of the full SystemC AMS class library as defined in this standard, only the public interface of which need be exposed to the application.

The word *application* is used to mean a C++ program, written by an end user, that uses the SystemC AMS class library, that is, it uses the classes, functions, or macros defined in this standard.

3.1.3 Call, called from, derived from

The term *call* is taken to mean call directly or indirectly. Call indirectly means call an intermediate function which in turn calls the function in question, where the chain of function calls may be extended indefinitely.

Similarly, *called from* means called from directly or indirectly.

Except where explicitly qualified, the term *derived from* is taken to mean derived directly or indirectly from. Derived indirectly from means derived from one or more intermediate base classes.

3.1.4 Specific technical terms

The specific technical terms as defined in IEEE Std 1666-2011 also apply for the AMS extensions. In addition, the following technical terms are defined:

A *model of computation* (MoC) is a set of rules defining the behavior and interaction between AMS primitive modules. The defined models of computation in this standard are: timed data flow (TDF), linear signal flow (LSF) and electrical linear networks (ELN).

A *cluster* is a set of AMS primitive modules connected by channels via AMS ports, which have no defined decoupling semantics. All modules have to be associated with the same model of computation.

An *AMS primitive module* is a class derived from the class **sca_core::sca_module** and associated with a model of computation. A primitive module cannot be hierarchically decomposed and contains no child modules or channels. A TDF module is a primitive module derived from class **sca_tdf::sca_module**. An LSF module is a primitive module derived from class **sca_lsf::sca_module**. An ELN module is a primitive module derived from class **sca_eln::sca_module**.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

An *AMS port* is a class derived from the class **sca_core::sca_port** and associated with a model of computation. A *primitive port* is a port of a primitive module.

An *AMS terminal* is a class derived from the class **sca_core::sca_port** and associated with the electrical linear networks model of computation.

An *AMS interface* is a class derived from the class **sca_core::sca_interface** and associated with a model of computation.

An *AMS interface proper* is an abstract class derived from the class **sca_core::sca_interface** and associated with a model of computation.

An *AMS channel* is a non-abstract class derived from one or more interfaces and associated with a model of computation.

An *AMS node* is an object of the class **sca_eln::sca_node** or **sca_eln::sca_node_ref** and is an AMS channel associated with the electrical linear networks model of computation.

An *AMS signal* is an object of the class **sca_tdf::sca_signal** or **sca_lsf::sca_signal** and is an AMS channel associated with the timed data flow or linear signal flow model of computation, respectively.

3.2 Syntactical conventions

3.2.1 Implementation-defined

The italicized term *implementation-defined* is used where part of a C++ definition is omitted from this standard. In such cases, an implementation shall provide an appropriate definition that honors the semantics defined in this standard.

3.2.2 Disabled

The italicized term *disabled* is used within a C++ class definition to indicate a group of member functions that shall be disabled by the implementation so that they cannot be called by an application. The disabled member functions are typically the default constructor, the copy constructor, or the assignment operator.

3.2.3 Ellipsis (...)

An ellipsis, which consists of three consecutive dots (...), is used to indicate that irrelevant or repetitive parts of a C++ code listing or example have been omitted for clarity.

3.2.4 Class names

Class names italicized and annotated with a superscript dagger (†) should not be used explicitly within an application. Moreover, an application shall not create an object of such a class. An implementation is strongly recommended to use the given class name. However, an implementation may substitute an alternative class name in place of every occurrence of a particular daggered class name.

Only the class name is being considered here. Whether any part of the definition of the class is implementation-defined is a separate issue.

The class names in question are the following:

sca_core::*sca_assign_from_proxy*†

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Full document details: BS IEC 61691-8:2021

sca_core::*sca_assign_to_proxy*†
sca_tdf::*sca_ct_proxy*†
sca_tdf::*sca_ct_vector_proxy*†
sca_util::*sca_information_mask*†
sca_util::*sca_traceable_object*†

3.2.5 Prefixes

The AMS extensions are denoted with the prefix **sca_** for namespaces, classes, functions, global definitions and variables and with the prefix **SCA_** for macros and enumeration values.

An application shall not make use of these prefixes for namespaces, functions, global definitions, variables, macros, and classes derived from the classes defined in this standard.

3.3 Typographical conventions

The following typographical conventions are used in this standard:

a) The *italic* font is used for:
   1) Cross references to terms defined in 3.1, 3.2, and Annex B.
   2) Arguments of member functions in class definitions and in the text that are generally substituted with real values by the implementation or application.

b) The **bold** font is used for all reserved keywords of SystemC and the AMS extensions as defined in namespaces, macros, constants, enum literals, classes, member functions, data members and types.

c) The `constant-width` (Courier) font is used:
   1) For the SystemC AMS class definition including its member functions, data members and data types.
   2) To illustrate SystemC AMS language examples when the exact usage is depicted.
   3) For references to the SystemC AMS language syntax and headers.

The conventions listed previously are for ease of reading only. Editorial inconsistencies in the use of typography are unintentional and have no normative meaning in this standard.

3.4 Semantic conventions

3.4.1 Class definitions and the inheritance hierarchy

An implementation may differ from this standard in that an implementation may introduce additional base classes, class members, and friends to the classes defined in this standard. An implementation may modify the inheritance hierarchy by moving class members defined by this standard into base classes not defined by this standard. Such additions and modifications may be made as necessary in order to implement the semantics defined by this standard or in order to introduce additional functionality not defined by this standard.

3.4.2 Function definitions and side-effects

This standard explicitly defines the semantics of the C++ functions for the AMS class library for SystemC. Such functions shall not have any side-effects that would contradict the behavior explicitly mandated by this standard. In general, the reader should assume the common-sense rule that if it is explicitly stated that a function shall perform action A, that function shall not perform any action other than A, either directly or by calling

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

another function defined in this standard. However, a function may, and indeed in certain circumstances shall, perform any tasks necessary for resource management, performance optimization, or to support any ancillary features of an implementation. As an example of resource management, it is presumed that a destructor will perform any tasks necessary to release the resources allocated by the corresponding constructor.

3.4.3 Functions whose return type is a reference or a pointer

Many functions in this standard return a reference to an object or a pointer to an object; that is, the return type of the function is a reference or a pointer. This subclause gives some general rules defining the lifetime and the validity of such objects.

An object returned from a function by pointer or by reference is said to be valid during any period in which the object is not deleted and the value or behavior of the object remains accessible to the application. If an application refers to the returned object after it ceases to be valid, the behavior of the implementation shall be undefined.

3.4.3.1 Functions that return *this or an actual argument

In certain cases, the object returned is either an object (***this**) returned by reference from its own member function (for example, the assignment operators) or is an object that was passed by reference as an actual argument to the function being called [for example, **std::ostream& operator<<(std::ostream&, const T&)**]. In either case, the function call itself places no additional obligations on the implementation concerning the lifetime and validity of the object following return from the function call.

3.4.3.2 Functions that return const char*

Certain functions have the return type **const char***; that is, they return a pointer to a null-terminated character string. The **const char*** pointer shall remain valid for the whole lifetime of the object associated to the member function, which returns this pointer.

3.4.4 Namespaces and internal naming

An implementation shall place every declaration and definition specified by this standard within one of the following namespaces: **sca_core**, **sca_tdf**, **sca_lsf**, **sca_eln**, **sca_ac_analysis**, or **sca_util**.

The core language base classes shall be placed in the namespace **sca_core**.

For the predefined models of computation, the following namespaces shall be used:

- The predefined classes for timed data flow shall be placed in the namespace **sca_tdf**.
- The predefined classes for linear signal flow shall be placed in the namespace **sca_lsf**.
- The predefined classes for electrical linear networks shall be placed in the namespace **sca_eln**.

The predefined classes for small-signal frequency-domain analyses shall be placed in the namespace **sca_ac_analysis**. The utilities shall be placed in the namespace **sca_util**.

It is recommended that an implementation uses nested namespaces within **sca_core**, **sca_tdf**, **sca_lsf**, **sca_eln**, **sca_ac_analysis**, and **sca_util** to reduce to a minimum the number of implementation-defined names in these namespaces.

For predefined primitive modules, which use ports to connect to a different model of computation, the namespace associated with the connected model of computation shall be used as nested namespace. The nested namespace **sca_de** shall be used for modules or ports, which are used to connect to SystemC discrete-event channels or ports.

Documents
Document Sharing Hub
Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Full document details: BS IEC 61691-8:2021

In general, the choice of internal, implementation-specific names within an implementation can cause naming conflicts within an application. It is up to the implementor to choose names that are unlikely to cause naming conflicts within an application.

3.4.5 Non-compliant applications and errors

In the case where an application fails to meet an obligation imposed by this standard, the behavior of the SystemC AMS implementation shall be undefined in general. When this results in the violation of a diagnosable rule of the C++ standard, the C++ implementation shall issue a diagnostic message in conformance with the C++ standard.

When this standard explicitly states that the failure of an application to meet a specific obligation is an *error* or a *warning*, the SystemC AMS implementation shall generate a diagnostic message by calling the function **sc_core::sc_report_handler::report**. In the case of an *error*, the implementation shall call function **report** with a severity of **sc_core::SC_ERROR**. In the case of a *warning*, the implementation shall call function **report** with a severity of **sc_core::SC_WARNING**.

An implementation or an application may choose to suppress run-time error checking and diagnostic messages because of considerations of efficiency or practicality. For example, an application may call member function **set_actions** of class **sc_core::sc_report_handler** to take no action for certain categories of reports. An application that fails to meet the obligations imposed by this standard remains in error. There are cases where this standard states explicitly that a certain behavior or result is *undefined*. This standard places no obligations on the implementation in such a circumstance. In particular, such a circumstance may or may not result in an *error* or a *warning* being issued.

3.5 Notes and examples

Notes appear at the end of certain subclauses, designated by the uppercase word NOTE. Notes often describe the consequences of rules defined elsewhere in this standard. Certain subclauses include examples consisting of fragments of C++ source code. Such notes and examples are informative to help the reader but are not an official part of this standard.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

4. Core language definitions

4.1 Class header files

To use the AMS class library features, an application shall include either of the C++ header files specified in this subclause at appropriate positions in the source code as required by the scope and linkage rules of C++.

4.1.1 #include “systemc-ams”

The header file named **systemc-ams** shall add the names **sca_core**, **sca_tdf**, **sca_lsf**, **sca_eln**, **sca_ac_analysis**, and **sca_util**, as well as the names defined in IEEE Std 1666-2011 for the header file named **systemc**, to the declarative region in which it is included. The header file **systemc-ams** shall not introduce into the declarative region, in which it is included, any other names from this standard or any names from the standard C or C++ libraries.

It is recommended that applications include the header file **systemc-ams** rather than the header file **systemc-ams.h**.

4.1.2 #include “systemc-ams.h”

The header file named **systemc-ams.h** shall add names from the namespace **sca_core, sca_ac_analysis**, **sca_util**, and **sca_tdf** as defined in this subclause to the declarative region, in which it is included. It is recommended that an implementation keeps the number of additional implementation-specific names introduced by this header file to a minimum.

The header file **systemc-ams.h** shall include at least the following:

```
#include "systemc.h"
#include "systemc-ams"

// Using declarations for the following names in the sca_ac_analysis namespace
using sca_ac_analysis::sca_ac_start;
using sca_ac_analysis::sca_ac_noise_start;
using sca_ac_analysis::sca_ac;
using sca_ac_analysis::sca_ac_is_running;
using sca_ac_analysis::sca_ac_noise;
using sca_ac_analysis::sca_ac_noise_is_running;
using sca_ac_analysis::sca_ac_f;
using sca_ac_analysis::sca_ac_w;
using sca_ac_analysis::sca_ac_s;
using sca_ac_analysis::sca_ac_z;
using sca_ac_analysis::sca_ac_delay;
using sca_ac_analysis::sca_ac_ltf_nd;
using sca_ac_analysis::sca_ac_ltf_zp;
using sca_ac_analysis::sca_ac_ss;
using sca_ac_analysis::SCA_LOG;
using sca_ac_analysis::SCA_LIN;

// Using declarations for the following names in the sca_util namespace
using sca_util::sca_trace_file;
using sca_util::sca_trace;
using sca_util::sca_create_tabular_trace_file;
using sca_util::sca_close_tabular_trace_file;
using sca_util::sca_create_vcd_trace_file;
using sca_util::sca_close_vcd_trace_file;
using sca_util::sca_write_comment;
using sca_util::sca_complex;
using sca_util::sca_matrix;
using sca_util::sca_vector;
using sca_util::sca_create_vector;
using sca_util::sca_information_on;
using sca_util::sca_information_off;
using sca_util::SCA_AC_REAL_IMAG;
```

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

```
using sca_util::SCA_AC_MAG_RAD;
using sca_util::SCA_AC_DB_DEG;
using sca_util::SCA_NOISE_SUM;
using sca_util::SCA_NOISE_ALL;
using sca_util::SCA_INTERPOLATE;
using sca_util::SCA_DONT_INTERPOLATE;
using sca_util::SCA_HOLD_SAMPLE;
using sca_util::sca_ac_format;
using sca_util::sca_noise_format;
using sca_util::sca_decimation;
using sca_util::sca_sampling;
using sca_util::sca_multirate;
using sca_util::SCA_COMPLEX_J;
using sca_util::SCA_INFINITY;
using sca_util::SCA_UNDEFINED;
namespace sca_info = sca_util::sca_info;

// Using declarations for the following names in the sca_core namespace
using sca_core::sca_parameter;
using sca_core::sca_time;
using sca_core::sca_max_time;
using sca_core::sca_copyright;
using sca_core::sca_version;
using sca_core::sca_release;

// Using declarations for the following names in the sca_tdf namespace
using sca_tdf::SCA_CT_CUT;
using sca_tdf::SCA_DT_CUT;
using sca_tdf::SCA_NO_CUT;
```

NOTE—The header file **systemc-ams.h** is provided for backward compatibility with earlier versions of the SystemC AMS extensions and may be deprecated in future versions of this standard.

4.2 Base class definitions

All names used in the base class definitions shall be placed in the namespace **sca_core**.

4.2.1 sca_core::sca_module

4.2.1.1 Description

The class **sca_core::sca_module** shall define the base class to derive primitive modules for the predefined models of computation.

4.2.1.2 Class definition

```
namespace sca_core {

  class sca_module : public sc_core::sc_module
  {
   public:
    virtual const char* kind() const;

    virtual void set_timestep( const sca_core::sca_time& );
    virtual void set_timestep( double, sc_core::sc_time_unit );
    virtual void set_max_timestep( const sca_core::sca_time& );
    virtual void set_max_timestep( double, sc_core::sc_time_unit );

   protected:
    sca_module();
    virtual ~sca_module();
  };

  #define SCA_CTOR(name)       implementation-defined name( sc_core::sc_module_name )

} // namespace sca_core
```

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

4.2.1.3 Constraints on usage

Any primitive module as defined in Clause 5, Clause 6 and Clause 7 shall be publicly derived from class **sca_core::sca_module**.

Objects of class **sca_core::sca_module** can only be constructed during elaboration. It shall be an error to instantiate a primitive module during simulation.

Although class **sca_core::sca_module** is derived from class **sc_core::sc_module**, the use of classes, member functions, and functions, which have direct access to the SystemC kernel shall not be allowed in the context of a module derived from class **sca_core::sca_module**.

The following member functions of class **sc_core::sc_module** shall not be called in the context of a module derived from class **sca_core::sca_module**:

a) All forms of member function **sc_core::sc_module::wait**.
b) All forms of member function **sc_core::sc_module::next_trigger**.
c) All forms of member function **sc_core::sc_module::reset_signal_is**.
d) All forms of member function **sc_core::sc_module::async_reset_signal_is**.
e) Member function **sc_core::sc_module::dont_initialize**.
f) Member function **sc_core::sc_module::set_stack_size**.

The following macros shall not be used in the context of a module derived from class **sca_core::sca_module**:

a) Macro **SC_CTOR**.
b) Macro **SC_HAS_PROCESS**.
c) Macro **SC_METHOD**.
d) Macro **SC_THREAD**.
e) Macro **SC_CTHREAD**.
f) Macro **SC_FORK**.
g) Macro **SC_JOIN**.

The following functions shall not be used in the context of a module derived from class **sca_core::sca_module**:

a) All forms of function **sc_core::wait**.
b) All forms of function **sc_core::next_trigger**.
c) Function **sc_core::sc_time_stamp**.
d) Function **sc_core::sc_delta_count**.
e) Function **sc_core::sc_get_current_process_handle**.
f) Function **sc_core::sc_spawn**.

Objects of the following classes shall not be created in the context of modules derived from class **sca_core::sca_module**:

a) Objects of class **sc_core::sc_event**.
b) Objects of class **sc_core::sc_process_handle**.
c) All objects which are derived from class **sc_core::sc_export_base**.
d) All objects which are derived from class **sc_core::sc_interface**.
e) All objects which are derived from class **sc_core::sc_module**.
f) All objects which are derived from class **sc_core::sc_port_base** and not from **sca_core::sca_port**.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

An application shall not derive from class **sca_core::sca_module** directly, but shall use the primitive modules defined in Clause 5, Clause 6, and Clause 7.

4.2.1.4 kind

```
virtual const char* kind() const;
```

The member function **kind** shall return the string "**sca_core::sca_module**".

4.2.1.5 set_timestep

```
virtual void set_timestep( const sca_core::sca_time& );

virtual void set_timestep( double, sc_core::sc_time_unit );
```

The member function **set_timestep** shall define the timestep of the module according to the execution semantics of the associated predefined model of computation (see 5.3, 6.3, 7.3). If the member function is not called, the current timestep of the module is computed as defined in the execution semantics of the associated predefined model of computation. It shall be an error to call this function after the first **sc_core::sc_module::end_of_elaboration** callback has been executed, except in the context of the callbacks **set_attributes** and **change_attributes** of the current module derived from class **sca_tdf::sca_module** (see 5.1.1.6).

4.2.1.6 set_max_timestep

```
virtual void set_max_timestep( const sca_core::sca_time& );

virtual void set_max_timestep( double, sc_core::sc_time_unit );
```

The member function **set_max_timestep** shall define the maximum timestep or the latest first activation time after elaboration of the module according to the execution semantics of the associated predefined model of computation (see 5.3, 6.3, 7.3). If **set_max_timestep** is not called, an implementation shall set the maximum timestep to the value returned by function **sca_core::sca_max_time**. It shall be an error to call this function after the first **sc_core::sc_module::end_of_elaboration** callback has been executed, except in the context of the callbacks **set_attributes** and **change_attributes** of the current module derived from class **sca_tdf::sca_module** (see 5.1.1.6).

NOTE—The propagated maximum timestep defined by the member function **set_max_timestep** is always respected.

4.2.1.7 SCA_CTOR

The macro **SCA_CTOR** is provided for convenience when declaring or defining a constructor for a module derived from class **sca_core::sca_module**. The macro shall only be used at a place where the rules of C++ permit a constructor to be declared and can be used as the declarator of a constructor declaration or a constructor definition. The name of the module class being constructed shall be passed as the argument to the macro.

4.2.2 sca_core::sca_interface

4.2.2.1 Description

The class **sca_core::sca_interface** shall define the base class for deriving interfaces for the predefined models of computation.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

4.2.2.2 Class definition

```
namespace sca_core {

  class sca_interface : public sc_core::sc_interface
  {
   protected:
    sca_interface();

   private:
    // Disabled
    sca_interface( const sca_core::sca_interface& );
    sca_core::sca_interface& operator= ( const sca_core::sca_interface& );
  };

} // namespace sca_core
```

4.2.2.3 Constraints on usage

An application shall not use class **sca_core::sca_interface** as the direct base class for any class other than an interface proper.

4.2.3 sca_core::sca_prim_channel

4.2.3.1 Description

The class **sca_core::sca_prim_channel** shall be used as base class to derive primitive channels for the predefined models of computation.

4.2.3.2 Class definition

```
namespace sca_core {

  class sca_prim_channel : public sc_core::sc_object,
                           public sca_util::sca_traceable_object†
  {
   public:
    virtual const char* kind() const;

   protected:
    sca_prim_channel();
    explicit sca_prim_channel( const char* );
    virtual ~sca_prim_channel();

   private:
    // Disabled
    sca_prim_channel( const sca_core::sca_prim_channel& );
    sca_core::sca_prim_channel& operator= ( const sca_core::sca_prim_channel& );
  };

} // namespace sca_core
```

4.2.3.3 Constraints on usage

Any primitive channel as defined in Clause 5, Clause 6, and Clause 7 shall be publicly derived from class **sca_core::sca_prim_channel**.

Objects of class **sca_core::sca_prim_channel** can only be constructed during elaboration. It shall be an error to instantiate a primitive channel during simulation.

NOTE—Since the constructors are protected, class **sca_core::sca_prim_channel** cannot be instantiated directly. An application shall use only the channels defined in Clause 5, Clause 6, and Clause 7 and shall not directly derive any channel from this class.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

4.2.3.4 Constructors

```
sca_prim_channel();

explicit sca_prim_channel( const char* );
```

The constructor for class **sca_core::sca_prim_channel** shall pass the character string argument (if such argument exists) through to the constructor belonging to the base class **sc_core::sc_object** to set the string name of the instance in the module hierarchy.

The default constructor shall call function **sc_core::sc_gen_unique_name**("**sca_prim_channel**") to generate a unique string name that it shall then pass through to the constructor belonging to the base class **sc_core::sc_object**.

4.2.3.5 kind

```
virtual const char* kind() const;
```

The member function **kind** shall return the string "**sca_core::sca_prim_channel**".

4.2.4 sca_core::sca_port

4.2.4.1 Description

The class **sca_core::sca_port** shall define the base class to derive ports for the predefined models of computation. It shall implement the interface of class **sca_util**::*sca_traceable_object*[†] in such a way that the channel, to which the port is bound, can be traced.

4.2.4.2 Class definition

```
namespace sca_core {

  template <class IF>
  class sca_port : public sc_core::sc_port<IF, 1, sc_core::SC_ONE_OR_MORE_BOUND >,
                   public sca_util::sca_traceable_object†
  {
   public:
    virtual const char* kind() const;

   protected:
    sca_port();
    explicit sca_port( const char* );
    virtual ~sca_port();
  };

} // namespace sca_core
```

4.2.4.3 Template parameter IF

The argument passed to template **sca_core::sca_port** shall be the name of an interface proper. The interface shall be derived from class **sc_core::sc_interface**.

4.2.4.4 Constraints on usage

An application shall not use class **sca_core::sca_port** to instantiate ports, but shall use the ports defined in Clause 5, Clause 6, and Clause 7.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

4.2.4.5 Constructors

```
sca_port();

explicit sca_port( const char* );
```

The constructor for class **sca_core::sca_port** shall pass the character string argument (if such argument exists) through to the constructor belonging to the base class **sc_core::sc_port** to set the string name of the instance in the module hierarchy.

The default constructor shall call function **sc_core::sc_gen_unique_name**(“**sca_port**”) to generate a unique string name that it shall then pass through to the constructor belonging to the base class **sc_core::sc_port**.

4.2.4.6 kind

```
virtual const char* kind() const;
```

The member function **kind** shall return the string “**sca_core::sca_port**”.

4.2.5 sca_core::sca_time

The class **sca_core::sca_time** shall be used to represent simulation time for the AMS extensions.

```
namespace sca_core { typedef sc_core::sc_time sca_time; }
```

NOTE 1—The typedef **sca_core::sca_time** has been introduced to facilitate future extensions to decouple the time resolution used in the AMS extensions from the time resolution as defined in IEEE Std 1666-2011.

NOTE 2—Since typedef **sca_core::sca_time** is an alias to **sc_core::sc_time**, an application can change the time resolution by calling the function **sc_core::sc_set_time_resolution** or query the time resolution by calling the function **sc_core::sc_get_time_resolution**.

4.2.6 sca_core::sca_max_time

The implementation shall provide a function **sca_core::sca_max_time** with the following declaration:

```
namespace sca_core { const sca_core::sca_time& sca_max_time(); }
```

The function **sca_core::sca_max_time** shall return the maximum value of type **sca_core::sca_time**, calculated after taking into account the time resolution. Since function **sca_core::sca_max_time** necessarily returns a reference to an object of type **sca_core::sca_time** that represents a non-zero time value, the time resolution cannot be modified after a call to **sca_core::sca_max_time**. Every call to **sca_core::sca_max_time** during a given simulation run shall return an object having the same value and representing the maximum simulation time. The actual value is implementation-defined. Whether each call to **sca_core::sca_max_time** returns a reference to the same object or a different object is implementation-defined.

4.2.7 sca_core::sca_parameter_base

4.2.7.1 Description

The class **sca_core::sca_parameter_base** shall define a type independent base class for module parameters. After construction, parameters shall be unlocked.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

NOTE—All instances of class **sca_core::sca_parameter_base** become part of the object hierarchy to facilitate access to the primitive module parameter values.

4.2.7.2 Class definition

```
namespace sca_core {

  class sca_parameter_base : public sc_core::sc_object
  {
   public:
    virtual const char* kind() const;

    virtual std::string to_string() const = 0 ;
    virtual void print( std::ostream& os = std::cout ) const = 0;

    void lock();
    void unlock();
    bool is_locked() const;

   protected:
    sca_parameter_base();
    explicit sca_parameter_base( const char* );
    virtual ~sca_parameter_base();

   private:
    // Disabled
    sca_parameter_base( const sca_core::sca_parameter_base& );
    sca_core::sca_parameter_base& operator= ( const sca_core::sca_parameter_base& );
  };

  std::ostream& operator<< ( std::ostream&, const sca_core::sca_parameter_base& );

} // namespace sca_core
```

4.2.7.3 Constructors

```
sca_parameter_base();

explicit sca_parameter_base( const char* );
```

The constructor for class **sca_core::sca_parameter_base** shall pass the character string argument (if such argument exists) through to the constructor belonging to the base class **sc_core::sc_object** to set the string name of the instance in the module hierarchy.

The default constructor shall call function **sc_core::sc_gen_unique_name**(“**sca_parameter_base**”) to generate a unique string name that it shall then pass through to the constructor belonging to the base class **sc_core::sc_object**.

4.2.7.4 kind

```
virtual const char* kind() const;
```

The member function **kind** shall return the string “**sca_core::sca_parameter_base**”.

4.2.7.5 to_string

```
virtual std::string to_string() = 0;
```

The member function **to_string** shall perform the conversion of the parameter value to an object of class std::string.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

4.2.7.6 print

```
virtual void print( std::ostream& os = std::cout ) = 0;
```

The member function **print** shall print the parameter value to the stream passed as an argument.

4.2.7.7 lock

```
void lock();
```

The member function **lock** shall prevent any further assignment to the parameter. It shall be an error if an assignment is executed on a locked parameter.

4.2.7.8 unlock

```
void unlock();
```

The member function **unlock** shall allow further assignments to the parameter.

4.2.7.9 is_locked

```
bool is_locked() const;
```

The member function **is_locked** shall return true if the parameter is locked; otherwise, it shall return false.

4.2.7.10 operator<<

```
std::ostream& operator<< ( std::ostream&, const sca_core::sca_parameter_base& );
```

The **operator<<** shall write the value of the parameter passed as the second argument to the stream passed as the first argument by calling the member function **print**(std::ostream).

4.2.8 sca_core::sca_parameter

4.2.8.1 Description

The class **sca_core::sca_parameter** shall assign a parameter to a module.

4.2.8.2 Class definition

```
namespace sca_core {

  template<class T>
  class sca_parameter : public sca_core::sca_parameter_base
  {
   public:
    sca_parameter();
    explicit sca_parameter( const char* name_ );
    sca_parameter( const char* name_, const T& default_value );
    ~sca_parameter();

    virtual const char* kind() const;

    virtual std::string to_string() const;
    virtual void print( std::ostream& os = std::cout ) const;
```

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

```
    const T& get() const;
    operator const T& () const;

    void set( const T& );
    sca_core::sca_parameter<T>& operator= ( const T& value );
    sca_core::sca_parameter<T>& operator= ( const sca_core::sca_parameter<T>& value );
  };

} // namespace sca_core
```

4.2.8.3 Template parameter T

The argument passed as template parameter **T** shall be either a C++ type for which the predefined semantics for assignment are adequate (for example, a fundamental type or a pointer) or a type **T** that obeys each of the following rules:

a) The following stream operator shall be defined and should copy the state of the object given as the second argument to the stream given as the first argument. The way in which the state information is formatted is not defined by this standard.

```
std::ostream& operator<< ( std::ostream&, const T& );
```

b) If the default assignment semantics are inadequate (in the sense given in this subclause), the following assignment operator shall be defined for the type **T**. In either case (default assignment or explicit operator), the semantics of assignment should be sufficient to assign the state of an object of type **T** such that the value of the left operand is indistinguishable from the value of the right operand.

```
const T& operator= ( const T& );
```

c) If any constructor for type **T** exists, a default constructor for type **T** shall be defined.

4.2.8.4 Constructors

The constructors shall only be called within the context of a **sc_core::sc_module** during module construction.

```
sca_parameter();
```

The default constructor shall call function **sc_core::sc_gen_unique_name**(“**sca_parameter**”) to generate a unique string name that it shall then pass through to the constructor belonging to the base class **sca_core::sca_parameter_base**. The actual parameter value shall be created by the default constructor of the corresponding type.

```
explicit sca_parameter( const char* name_ );
```

The constructor shall pass the character string *name_* through to the constructor belonging to the base class **sca_core::sca_parameter_base** to set the string name of the instance in the module hierarchy. The actual parameter value shall be created by the default constructor of the corresponding type.

```
sca_parameter( const char* name_, const T& default_value );
```

The constructor shall pass the character string *name_* through to the constructor belonging to the base class **sca_core::sca_parameter_base** to set the string name of the instance in the module hierarchy. The actual parameter value shall be created by the default constructor and initialized with the default value *default_value*.

4.2.8.5 kind

```
virtual const char* kind() const;
```

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

The member function **kind** shall return the string "**sca_core::sca_parameter**".

4.2.8.6 to_string

```
virtual std::string to_string() const;
```

The member function **to_string** shall perform the conversion of the parameter value to an object of class std::string. Conversion shall be done by calling **operator<<** (std::ostream&, const T&). (See 4.2.8.3.)

4.2.8.7 print

```
virtual void print( std::ostream& os = std::cout ) const;
```

The member function **print** shall print the current parameter value to the stream passed as an argument by calling **operator<<** (std::ostream&, const T&). (See 4.2.8.3.)

4.2.8.8 get

```
const T& get() const;

operator const T& () const;
```

The member function **get** and **operator const T&** shall return a const reference to the actual parameter value. If the member functions **lock** or **unlock** have not yet executed, the member function **get** and **operator const T&** shall execute the member function **lock** of the base class **sca_core::sca_parameter_base**.

4.2.8.9 set

```
void set( const T& value );

sca_core::sca_parameter<T>& operator= ( const T& value );
```

The member function **set** and **operator=** shall assign the *value* to the parameter. It shall be an error if the member function **set** or **operator=** is called if the parameter is locked (see 4.2.7.9).

```
sca_core::sca_parameter<T>& operator= ( const sca_core::sca_parameter<T>& value );
```

The **operator=** shall copy the value of the parameter passed as argument.

4.2.9 sca_core::*sca_assign_from_proxy*†

4.2.9.1 Description

The class **sca_core::***sca_assign_from_proxy*† shall be a helper class to facilitate the implementation of the assignment operator from one class to another.

NOTE—This class is the base class of **sca_tdf::***sca_ct_proxy*† and **sca_tdf::***sca_ct_vector_proxy*† to implement the assignment of the values returned by these classes to a port or vector, whose type is passed to this base class as template parameter of type **T**.

4.2.9.2 Class definition

```
namespace sca_core {
```

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

```
  template<class T>
  class sca_assign_from_proxy†
  {
    implementation-defined
  };

} // namespace sca_core
```

4.2.9.3 Constraint on usage

An application shall not explicitly create an instance of class **sca_core::***sca_assign_from_proxy*†.

4.2.10 sca_core::*sca_assign_to_proxy*†

4.2.10.1 Description

The class **sca_core::***sca_assign_to_proxy*† shall be a helper class to facilitate the implementation of operators to assign values to an object.

NOTE—This class is used to support the implementation of the **operator[]** for the classes **sca_tdf::sca_out** and **sca_tdf::sca_de::sca_out**.

4.2.10.2 Class definition

```
namespace sca_core {

  template<class T, class TV>
  class sca_assign_to_proxy†
  {
    sca_core::sca_assign_to_proxy†<T, TV>& operator= ( const TV& value );
  };

} // namespace sca_core
```

4.2.10.3 operator=

```
sca_assign_to_proxy†<T, TV>& operator= ( const TV& value );
```

The **operator=** performs the assignment of value *value* to an object of class **T**.

4.2.10.4 Constraint on usage

An application shall not explicitly create an instance of class **sca_core::***sca_assign_to_proxy*†.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

5. Timed data flow model of computation

The TDF model of computation shall define the procedural behavior that processes samples, which are tagged in time. A TDF module shall define time domain processing, which is activated when a predefined number of samples is available at its input port(s), and generates a predefined number of output samples at its output port(s). Since the number of read and written samples is known and fixed, the activation schedule of a set of connected TDF modules can be statically determined. For the communication with the SystemC kernel, predefined specialized ports shall be used to maintain synchronization. For synchronization, the tagged time of the samples shall be used. A TDF module is a primitive module that cannot be further hierarchically decomposed.

5.1 Class definitions

All names used in the TDF class definitions shall be placed in the namespace **sca_tdf**.

5.1.1 sca_tdf::sca_module

5.1.1.1 Description

The class **sca_tdf::sca_module** shall define the base class for all TDF primitive modules.

5.1.1.2 Class definition

```
namespace sca_tdf {

  class sca_module : public sca_core::sca_module
  {
   public:
    virtual const char* kind() const;

   protected:
    typedef void ( sca_tdf::sca_module::*sca_module_method )();

    virtual void set_attributes();
    virtual void change_attributes();
    virtual void initialize();
    virtual void reinitialize();
    virtual void processing();
    virtual void ac_processing();

    void register_processing( sca_tdf::sca_module::sca_module_method );
    void register_ac_processing( sca_tdf::sca_module::sca_module_method );

    void request_next_activation( const sca_core::sca_time& );
    void request_next_activation( double, sc_core::sc_time_unit );
    void request_next_activation( const sc_core::sc_event& );
    void request_next_activation( const sca_core::sca_time&, const sc_core::sc_event& );
    void request_next_activation( double, sc_core::sc_time_unit, const sc_core::sc_event& );
    void request_next_activation( const sc_core::sc_event_or_list& );
    void request_next_activation( const sc_core::sc_event_and_list& );
    void request_next_activation( const sca_core::sca_time&, const sc_core::sc_event_or_list& );
    void request_next_activation( double, sc_core::sc_time_unit, const sc_core::sc_event_or_list& );
    void request_next_activation( const sca_core::sca_time&, const sc_core::sc_event_and_list& );
    void request_next_activation( double, sc_core::sc_time_unit, const sc_core::sc_event_and_list& );

    template<class T>
    void request_next_activation( const sca_tdf::sca_de::sca_in<T>& );

    void accept_attribute_changes();
    void reject_attribute_changes();
    void does_attribute_changes();
    void does_no_attribute_changes();

    sca_core::sca_time get_time() const;
```

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

```
    sca_core::sca_time get_timestep() const;
    sca_core::sca_time get_max_timestep() const;
    sca_core::sca_time get_last_timestep() const;

    bool is_dynamic() const;
    bool are_attribute_changes_allowed() const;
    bool are_attributes_changed() const;
    bool is_timestep_changed() const;

    explicit sca_module( const sc_core::sc_module_name& );
    sca_module();

    virtual ~sca_module();
  };

#define SCA_TDF_MODULE(name) struct name : sca_tdf::sca_module

} // namespace sca_tdf
```

5.1.1.3 Constraints on usage

Modules, channels, signals, and ports outside of the namespace **sca_tdf** shall not be instantiated in the context of class **sca_tdf::sca_module**.

Objects of class **sca_tdf::sca_module** can only be constructed during elaboration. It shall be an error to instantiate such a module during simulation. Every class derived (directly or indirectly) from class **sca_tdf::sca_module** shall have at least one constructor. Every such constructor shall have one and only one argument of class **sc_core::sc_module_name**. A string-valued argument shall be passed to the constructor of every module instance.

Inter-module communication for TDF modules shall be accomplished using interface method calls, that is, a module should communicate with its environment through its ports.

5.1.1.4 kind

```
virtual const char* kind() const;
```

The member function **kind** shall return the string "**sca_tdf::sca_module**".

5.1.1.5 set_attributes

```
virtual void set_attributes();
```

The member function **set_attributes** shall provide a context to set attributes, which are required for TDF MoC elaboration (see 5.3). The attributes can be defined using the member functions **set_timestep**, **set_max_timestep**, and **request_next_activation** of a TDF module and member functions **set_timestep**, **set_max_timestep**, **set_delay**, and **set_rate** for ports of classes **sca_tdf::sca_in**, **sca_tdf::sca_out**, **sca_tdf::sca_de::sca_in**, **sca_tdf::sca_de::sca_out**, and in addition **set_ct_delay** for ports of classes **sca_tdf::sca_out**<T, **sca_tdf::SCA_CT_CUT**, INTERP> and **sca_tdf::sca_out**<T, **sca_tdf::SCA_DT_CUT**>. The member function **set_attributes** shall be called during the elaboration phase (see 5.3.1.1). The application shall not call this member function.

5.1.1.6 change_attributes

```
virtual void change_attributes();
```

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

The member function **change_attributes** shall provide a context to change attributes of the TDF module and its ports. The attributes can be changed using the member functions **set_timestep**, **set_max_timestep**, or **request_next_activation** of a TDF module and member functions **set_timestep**, **set_max_timestep**, **set_delay**, and **set_rate** for ports of classes **sca_tdf::sca_in**, **sca_tdf::sca_out**, **sca_tdf::sca_de::sca_in**, **sca_tdf::sca_de::sca_out**, and in addition **set_ct_delay** for ports of classes **sca_tdf::sca_out**<T, **sca_tdf::SCA_CT_CUT**, INTERP> and **sca_tdf::sca_out**<T, **sca_tdf::SCA_DT_CUT**>. The member function shall only be called during simulation (see 5.3.2). The application shall not call this member function.

5.1.1.7 initialize

```
virtual void initialize();
```

The member function **initialize** shall provide a context to set initial values to member variables and ports. In the context of this member function, the application can initialize the delay samples of all ports if their delay attribute has been set to a value greater than zero by using member function **initialize** of ports of classes **sca_tdf::sca_in**, **sca_tdf::sca_out**, **sca_tdf::sca_de::sca_in**, and **sca_tdf::sca_de::sca_out**. The member function shall only be called during simulation (see 5.3.2.1). The application shall not call this member function.

5.1.1.8 reinitialize

```
virtual void reinitialize();
```

The member function **reinitialize** shall provide a context to reinitialize values to member variables and ports. The member function shall be called after each cluster execution period (see 5.3.2.3). The application shall not call this member function.

NOTE—This member function may be used to reinitialize the delay samples of the associated ports by using the member function **initialize** in case of attribute changes.

5.1.1.9 processing

```
virtual void processing();
```

The member function **processing** shall provide a context to define the time-domain behavior of the TDF module. It may be replaced by a registered application-defined member function (see 5.1.1.11). It shall be a warning if a TDF module does not implement a single member function **processing** or a registered application-defined member function when time-domain simulation starts. If no application-defined member function is registered, this member function shall be called during time-domain simulation (see 8.1). The application shall not call this member function.

5.1.1.10 ac_processing

```
virtual void ac_processing();
```

The member function **ac_processing** shall provide a context to define the small-signal frequency-domain behavior of the TDF module. It may be replaced by a registered application-defined member function (see 5.1.1.12). If no application-defined member function is registered, this function shall be called during small-signal frequency-domain simulation (see 8.2). The application shall not call this member function.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

5.1.1.11 register_processing

```
void register_processing( sca_tdf::sca_module::sca_module_method );
```

The member function **register_processing** shall register a time-domain processing member function as a replacement to the default time-domain processing member function **processing**. The argument shall be a pointer to a member function of the TDF module. The registered application-defined member function shall behave in the same way as defined in member function **processing** (see 5.1.1.9). The member function **register_processing** shall only be called during module construction; otherwise, it shall be an error. It shall be an error if more than one member function is registered.

5.1.1.12 register_ac_processing

```
void register_ac_processing( sca_tdf::sca_module::sca_module_method );
```

The member function **register_ac_processing** shall register a small-signal frequency-domain processing member function as a replacement to the default small-signal frequency-domain processing member function **ac_processing**. The argument shall be a pointer to a member function of the TDF module. The registered application-defined member function shall behave in the same way as defined in member function **ac_processing** (see 5.1.1.10). The member function **register_ac_processing** shall only be called during module construction; otherwise, it shall be an error. It shall be an error if more than one member function is registered.

5.1.1.13 request_next_activation

The member function **request_next_activation** shall override the propagated timestep defined by the member function **set_timestep** of the TDF modules and ports for the next module activation. The next module activation shall be no later than the time requested by this member function.

The time given as argument shall be taken to be relative to the time in the context of the member function **change_attributes** of the current module, in which the member function **request_next_activation** is called.

```
void request_next_activation( const sca_core::sca_time& );

void request_next_activation( double, sc_core::sc_time_unit );
```

The next module activation is requested after the time given as an argument has elapsed.

```
void request_next_activation( const sc_core::sc_event& );
```

The next module activation is requested when the event, passed as an argument, is notified.

```
void request_next_activation( const sca_core::sca_time&, const sc_core::sc_event& );

void request_next_activation( double, sc_core::sc_time_unit, const sc_core::sc_event& );
```

The next module activation is requested after the time given as an argument has elapsed or when the given event is notified, whichever occurs first.

```
void request_next_activation( const sc_core::sc_event_or_list& );

void request_next_activation( const sc_core::sc_event_and_list& );
```

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

The next module activation is requested based on the event list, passed as argument.

```
void request_next_activation( const sca_core::sca_time&, const sc_core::sc_event_or_list& );

void request_next_activation( double, sc_core::sc_time_unit, const sc_core::sc_event_or_list& );


void request_next_activation( const sca_core::sca_time&, const sc_core::sc_event_and_list& );

void request_next_activation( double, sc_core::sc_time_unit, const sc_core::sc_event_and_list& );
```

The next module activation is requested after the time given as an argument has elapsed or in response to the event list, passed as argument, whichever is satisfied first.

```
template<class T>
void request_next_activation( const sca_tdf::sca_de::sca_in<T>& );
```

The next module activation is requested when the event, which is returned by the member function **default_event** of the port, passed as an argument, is notified.

5.1.1.14 accept_attribute_changes

```
void accept_attribute_changes();
```

The member function **accept_attribute_changes** shall mark a TDF module to accept attribute changes caused by other TDF modules, which belong to the same TDF cluster, after all **set_attributes** or **change_attributes** callbacks of the current cluster execution have been executed (see 5.3.2.3). It shall be an error if the member function is called outside the context of the member functions **set_attributes**, **change_attributes**, or the constructor of the current TDF module. If this member function is not called, a TDF module shall not accept attribute changes.

NOTE—A TDF module which accepts attribute changes is not allowed to change attributes itself, unless the member function **does_attribute_changes** has been called.

5.1.1.15 reject_attribute_changes

```
void reject_attribute_changes();
```

The member function **reject_attribute_changes** shall mark a TDF module to reject attribute changes made by other TDF modules, which belong to the same TDF cluster, after all **set_attributes** or **change_attributes** callbacks of the current cluster execution have been executed (see 5.3.2.3). It shall be an error if the TDF module is marked to reject attribute changes, and if other TDF modules, which belong to the same TDF cluster, change the attributes. It shall be an error if the member function is called outside the context of the member functions **set_attributes**, **change_attributes**, or the constructor of the current TDF module. If this member function is not called, a TDF module shall reject attribute changes.

NOTE—A TDF module which rejects attribute changes may still change attributes itself, as long as the member function **does_attribute_changes** has been called.

5.1.1.16 does_attribute_changes

```
void does_attribute_changes();
```

The member function **does_attribute_changes** shall mark a TDF module to allow it to make attribute changes after all **set_attributes** or **change_attributes** callbacks of the current cluster execution have been executed

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

(see 5.3.2.3). It shall be a warning if the TDF module is marked to allow making attribute changes and if there is no **change_attributes** callback implemented in the TDF module. It shall be an error if the member function is called outside the context of the member functions **set_attributes**, **change_attributes**, or the constructor of the current TDF module. If this member function is not called, a TDF module is not allowed to make attribute changes.

5.1.1.17 does_no_attribute_changes

```
void does_no_attribute_changes();
```

The member function **does_no_attribute_changes** shall mark a TDF module to disallow it to make attribute changes after all **set_attributes** or **change_attributes** callbacks of the current cluster execution have been executed (see 5.3.2.3). It shall be an error if the TDF module is marked to disallow making attribute changes, but changes its attributes or the ones of its ports. It shall be an error if the member function is called outside the context of the member functions **set_attributes**, **change_attributes**, or the constructor of the current TDF module. If this member function is not called, a TDF module is not allowed to make attribute changes.

5.1.1.18 get_time

```
sca_core::sca_time get_time() const;
```

The member function **get_time** shall return the current module time of type **sca_core::sca_time**. It represents the time of the first input sample of the current module activation. It shall be an error if the member function is called outside the context of the member functions **initialize**, **reinitialize**, **processing**, **ac_processing**, or **change_attributes** of the current TDF module.

NOTE—The function **sc_core::sc_time_stamp** should not be used in a TDF module as there may be time offsets between the current module time of the TDF module and the SystemC kernel time.

5.1.1.19 get_timestep

```
sca_core::sca_time get_timestep() const;
```

The member function **get_timestep** shall return the current timestep of the module according to the execution semantics (see 5.3). It shall be an error if the member function is called outside the context of the member functions **initialize**, **reinitialize**, **processing**, **ac_processing**, or **change_attributes** of the current TDF module.

5.1.1.20 get_max_timestep

```
sca_core::sca_time get_max_timestep() const;
```

The member function **get_max_timestep** shall return the maximum timestep of the module according to the execution semantics (see 5.3), ignoring the timesteps set for all TDF modules and TDF ports through the member function **set_timestep**. It shall be an error if the member function is called outside the context of the member functions **initialize**, **reinitialize**, **processing**, **ac_processing**, or **change_attributes** of the current TDF module.

NOTE—The member function **request_next_activation** overrules the resolved timestep set by the member functions **set_timestep** of the TDF modules and needs to satisfy the resolved maximum timestep set by member functions **set_max_timestep** of the TDF modules. The member function **get_max_timestep** returns the maximum resolved timestep.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

5.1.1.21 get_last_timestep

```
sca_core::sca_time get_last_timestep() const;
```

The member function **get_last_timestep** shall return the last non-zero module timestep of type **sca_core::sca_time** of the last module activation or before. For the first module activation, the member function shall return the propagated timestep. It shall be an error if the member function is called outside the context of the member functions **initialize**, **reinitialize**, **processing**, **ac_processing**, or **change_attributes** of the current TDF module.

5.1.1.22 is_dynamic

```
bool is_dynamic() const;
```

The member function **is_dynamic** shall return true if at least one TDF module within the TDF cluster has been marked to allow making attribute changes using the member function **does_attribute_changes**. Otherwise, it shall return false. The state returned by the member function **is_dynamic** is updated after the execution of all member functions **change_attributes** of the current cluster (see 5.3.2.3). It shall be an error if the member function is called outside the context of the member functions **initialize**, **reinitialize**, **processing**, **ac_processing**, or **change_attributes** of the current TDF module.

5.1.1.23 are_attribute_changes_allowed

```
bool are_attribute_changes_allowed() const;
```

The member function **are_attribute_changes_allowed** shall return true if changes to the attributes of the TDF module, in which the member function is called, and its ports are allowed. Otherwise, it shall return false. It shall be an error if the member function is called outside the context of the member functions **initialize**, **reinitialize**, **processing**, **ac_processing**, or **change_attributes** of the current TDF module. By default, a TDF module is not allowed to make changes to its attributes.

5.1.1.24 are_attributes_changed

```
bool are_attributes_changed() const;
```

In the context of the callbacks **processing**, **ac_processing**, and **reinitialize**, the member function **are_attributes_changed** shall return true if the timestep, delay, or rate of the TDF module, or its ports, have changed since the last activation of the callback **processing**. In the context of the callback **change_attributes**, the member function **are_attributes_changed** shall return true if the timestep, delay, or rate of the TDF module, or its ports, have changed between the last activation and before last activation of the callback **processing**. Otherwise, it shall return false. It shall be an error if the member function is called outside the context of the member functions **initialize**, **reinitialize**, **processing**, **ac_processing**, or **change_attributes** of the current TDF module.

5.1.1.25 is_timestep_changed

```
bool is_timestep_changed() const;
```

In the context of the callbacks **processing**, **ac_processing**, and **reinitialize**, the member function **is_timestep_changed** shall return true if the timestep of the TDF module, or its ports, have changed since the last activation of the callback **processing**. In the context of the callback **change_attributes**, the member function **is_timestep_changed** shall return true if the timestep of the TDF module, or its ports, have changed

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

between the last activation and before last activation of the callback **processing**. Otherwise, it shall return false. It shall be an error if the member function is called outside the context of the member functions **initialize**, **reinitialize**, **processing**, **ac_processing**, or **change_attributes** of the current TDF module.

NOTE—A change in the propagated timestep (see 5.3.1.2) may change the TDF module or port timestep, which is then also detected by this member function.

5.1.1.26 Constructor

```
explicit sca_module( const sc_core::sc_module_name& );

sca_module();
```

Module names are managed by class **sc_core::sc_module_name**, not by class **sca_tdf::sca_module**. The string name of the module instance is initialized using the value of the string name passed as an argument to the constructor.

5.1.1.27 SCA_TDF_MODULE

The macro **SCA_TDF_MODULE** may be used to prefix the definition of a **sca_tdf::sca_module**, but the use of the macro is not obligatory.

Example:

```
SCA_TDF_MODULE(M1)
{
  // ports, data members, member functions
  ...

  SCA_CTOR(M1);
};

M1::M1( sc_core::sc_module_name )
{
  // constructor body
}
```

5.1.2 sca_tdf::sca_signal_if

5.1.2.1 Description

The class **sca_tdf::sca_signal_if** shall define an interface proper for a primitive channel of class **sca_tdf::sca_signal**. The interface class member functions are implementation-defined.

5.1.2.2 Class definition

```
namespace sca_tdf {

  template<class T>
  class sca_signal_if : public sca_core::sca_interface
  {
   protected:
    sca_signal_if();

   private:
    // Other members
    implementation-defined

    // Disabled
    sca_signal_if( const sca_tdf::sca_signal_if<T>& );
    sca_tdf::sca_signal_if<T>& operator= ( const sca_tdf::sca_signal_if<T>& );
```

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

```
  };

} // namespace sca_tdf
```

5.1.3 sca_tdf::sca_signal

5.1.3.1 Description

The class **sca_tdf::sca_signal** shall define a primitive channel for the TDF MoC. It shall be used for connecting modules derived from class **sca_tdf::sca_module** using port classes **sca_tdf::sca_in** and **sca_tdf::sca_out**. An application shall not access the associated interface directly.

5.1.3.2 Class definition

```
namespace sca_tdf {

  template<class T>
  class sca_signal : public sca_tdf::sca_signal_if<T>,
                     public sca_core::sca_prim_channel
  {
   public:
    sca_signal();
    explicit sca_signal( const char* );

    virtual const char* kind() const;

   private:
    // Disabled
    sca_signal( const sca_tdf::sca_signal<T>& );
  };

} // namespace sca_tdf
```

5.1.3.3 Template parameter T

The argument passed as template parameter **T** shall be either a C++ type for which the predefined semantics for assignment are adequate (for example, a fundamental type or a pointer) or a type **T** that obeys each of the following rules:

a) The following stream operator shall be defined and should copy the state of the object given as the second argument to the stream given as the first argument. The way in which the state information is formatted is not defined by this standard. The implementation shall use this operator for writing trace values in time-domain simulation (see 9.1).

```
std::ostream& operator<< ( std::ostream&, const T& );
```

b) If the default assignment semantics are inadequate (in the sense given in this subclause), the following assignment operator shall be defined for the type **T**. In either case (default assignment or explicit operator), the semantics of assignment should be sufficient to assign the state of an object of type **T** such that the value of the left operand is indistinguishable from the value of the right operand. The implementation shall use this assignment operator within the implementation for writing to or reading from ports of type **T**.

```
const T& operator= ( const T& );
```

c) If any constructor for type **T** exists, a default constructor for type **T** shall be defined.

5.1.3.4 Constructors

```
sca_signal();

explicit sca_signal( const char* );
```

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

The constructor for class **sca_tdf::sca_signal** shall pass the character string argument (if such argument exists) through to the constructor belonging to the base class **sca_core::sca_prim_channel** to set the string name of the instance in the module hierarchy.

The default constructor shall call function **sc_core::sc_gen_unique_name**("**sca_tdf_signal**") to generate a unique string name that it shall then pass through to the constructor belonging to the base class **sca_core::sca_prim_channel**.

5.1.3.5 kind

```
virtual const char* kind() const;
```

The member function **kind** shall return the string "**sca_tdf::sca_signal**".

5.1.4 sca_tdf::sca_default_interpolator

5.1.4.1 Description

The class **sca_tdf::sca_default_interpolator** shall define the default interpolation mechanism for the continuous-time decoupling port of class **sca_tdf::sca_out**<T, **sca_tdf::SCA_CT_CUT**, INTERP>. The specialized classes **sca_tdf::sca_default_interpolator**<double> and **sca_tdf::sca_default_interpolator**<**sca_util::sca_complex**> shall provide a default interpolation mechanism by interpreting the signal as continuous in time. For all other types, the class shall keep the value of the last available time point.

5.1.4.2 Class definition

```
namespace sca_tdf {

  template<class T>
  class sca_default_interpolator
  {
   public:
    void store_value( const sca_core::sca_time&, const T& );
    T get_value( const sca_core::sca_time& ) const;
  };

  template<>
  class sca_default_interpolator<double>
  {
   public:
    void store_value( const sca_core::sca_time&, const double& );
    double get_value( const sca_core::sca_time& ) const;
  };

  template<>
  class sca_default_interpolator<sca_util::sca_complex>
  {
   public:
    void store_value( const sca_core::sca_time&, const sca_util::sca_complex& );
    sca_util::sca_complex get_value( const sca_core::sca_time& ) const;
  };

} // namespace sca_tdf
```

5.1.4.3 Template parameter T

The argument passed as template parameter **T** shall be either a C++ type for which the predefined semantics for assignment are adequate (for example, a fundamental type or a pointer) or a type **T** that obeys each of the following rules:

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

a) If the default assignment semantics are inadequate (in the sense given in this subclause), the following assignment operator shall be defined for the type **T**. In either case (default assignment or explicit operator), the semantics of assignment should be sufficient to assign the state of an object of type **T** such that the value of the left operand is indistinguishable from the value of the right operand.

```
const T& operator= ( const T& );
```

b) If any constructor for type **T** exists, a default constructor for type **T** shall be defined.

5.1.4.4 store_value

```
void store_value( const sca_core::sca_time&, const T& );

void store_value( const sca_core::sca_time&, const double& );

void store_value( const sca_core::sca_time&, const sca_util::sca_complex& );
```

An implementation shall use the member function **store_value** to store a time-value pair to the interpolator class.

5.1.4.5 get_value

```
T get_value( const sca_core::sca_time& ) const;

double get_value( const sca_core::sca_time& ) const;

sca_util::sca_complex get_value( const sca_core::sca_time& ) const;
```

An implementation shall use the member function **get_value** to return an interpolated value at a given time point given as argument.

5.1.5 sca_tdf::sca_in

5.1.5.1 Description

The class **sca_tdf::sca_in** shall define a port class for the TDF MoC. It provides functions for defining or getting attribute values (e.g., sampling rate or timestep), for initialization, and for reading input samples.

5.1.5.2 Class definition

```
namespace sca_tdf {

  template<class T>
  class sca_in : public sca_core::sca_port< sca_tdf::sca_signal_if<T> >
  {
   public:
    sca_in();
    explicit sca_in( const char* );

    void set_delay( unsigned long );
    void set_rate( unsigned long );
    void set_timestep( const sca_core::sca_time& );
    void set_timestep( double, sc_core::sc_time_unit );
    void set_max_timestep( const sca_core::sca_time& );
    void set_max_timestep( double, sc_core::sc_time_unit );

    unsigned long get_delay() const;
    unsigned long get_rate() const;
    sca_core::sca_time get_time( unsigned long sample_id = 0 ) const;
    sca_core::sca_time get_timestep( unsigned long sample_id = 0 ) const;
    sca_core::sca_time get_max_timestep() const;
    sca_core::sca_time get_last_timestep( unsigned long sample_id = 0 ) const;
```

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61691-8:2021

```
    virtual const char* kind() const;

    void initialize( const T& value, unsigned long sample_id = 0 );
    const T& read_delayed_value( unsigned long sample_id = 0 ) const;

    bool is_timestep_changed( unsigned long sample_id = 0 ) const;
    bool is_rate_changed() const;
    bool is_delay_changed() const;

    const T& read( unsigned long sample_id = 0 ) const;
    operator const T& () const;
    const T& operator[] ( unsigned long sample_id ) const;

   private:
    // Disabled
    sca_in( const sca_tdf::sca_in<T>& );
    sca_tdf::sca_in<T>& operator= ( const sca_tdf::sca_in<T>& );
  };

} // namespace sca_tdf
```

5.1.5.3 Template parameter T

The argument passed as template parameter **T** shall be either a C++ type for which the predefined semantics for assignment are adequate (for example, a fundamental type or a pointer) or a type **T** that obeys each of the following rules:

a) The following stream operator shall be defined and should copy the state of the object given as the second argument to the stream given as the first argument. The way in which the state information is formatted is not defined by this standard. The implementation shall use this operator for writing trace values in time-domain simulation (see 9.1).

```
std::ostream& operator<< ( std::ostream&, const T& );
```

b) If the default assignment semantics are inadequate (in the sense given in this subclause), the following assignment operator shall be defined for the type **T**. In either case (default assignment or explicit operator), the semantics of assignment should be sufficient to assign the state of an object of type **T** such that the value of the left operand is indistinguishable from the value of the right operand. The implementation shall use this assignment operator within the implementation for writing to or reading from ports of type **T**.

```
const T& operator= ( const T& );
```

c) If any constructor for type **T** exists, a default constructor for type **T** shall be defined.

5.1.5.4 Constructors

```
sca_in();

explicit sca_in( const char* );
```

The constructor for class **sca_tdf::sca_in** shall pass the character string argument (if such argument exists) through to the constructor belonging to the base class **sca_core::sca_port** to set the string name of the instance in the module hierarchy.

The default constructor shall call function **sc_core::sc_gen_unique_name**(“**sca_tdf_in**”) to generate a unique string name that it shall then pass through to the constructor belonging to the base class **sca_core::sca_port**.

5.1.5.5 set_delay

```
void set_delay( unsigned long );
```

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021

The member function **set_delay** shall define the number of samples to be inserted before the first input sample. If the member function is not called, the port shall have a delay of zero. It shall be an error if the member function is called outside the context of the member functions **set_attributes** or **change_attributes** of the current TDF module.

5.1.5.6 set_rate

```
void set_rate( unsigned long );
```

The member function **set_rate** shall define the number of samples that can be read during the execution of the member function **processing** of the current TDF module by using member function **read**. The argument *rate* shall have a positive, nonzero value. If the member function is not called, the port rate shall be equal to 1. It shall be an error if the member function is called outside the context of the member functions **set_attributes** or **change_attributes** of the current TDF module.

5.1.5.7 set_timestep

```
void set_timestep( const sca_core::sca_time& );

void set_timestep( double, sc_core::sc_time_unit );
```

The member function **set_timestep** shall define the timestep between two consecutive samples. If the member function is not called, the current timestep of the port is computed as defined in the execution semantics (see 5.3). It shall be an error if the member function is called outside the context of the member functions **set_attributes** or **change_attributes** of the current TDF module.

5.1.5.8 set_max_timestep

```
void set_max_timestep( const sca_core::sca_time& );

void set_max_timestep( double, sc_core::sc_time_unit );
```

The member function **set_max_timestep** shall define the maximum timestep between two consecutive samples. If **set_max_timestep** is not called, an implementation shall set the maximum timestep to the value returned by function **sca_core::sca_max_time**. It shall be an error if the member function is called outside the context of the member functions **set_attributes** or **change_attributes** of the current TDF module.

5.1.5.9 get_delay

```
unsigned long get_delay() const;
```

The member function **get_delay** shall return the delay set at the port. It shall be an error if the member function is called outside the context of the member functions **initialize**, **reinitialize**, **processing**, **ac_processing**, or **change_attributes** of the current TDF module.

5.1.5.10 get_rate

```
unsigned long get_rate() const;
```

The member function **get_rate** shall return the rate set at the port. It shall be an error if the member function is called outside the context of the member functions **initialize**, **reinitialize**, **processing**, **ac_processing**, or **change_attributes** of the current TDF module.

Published by IEC under licence from IEEE. © 2016 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61691-8:2021